

Teoría de lenguajes

Universidad de Buenos Aires
Departamento de computación

Notas de estudio

Por: Raul Latashen (+ partes de la carpeta de Soledad Ramusio)

Esquema de temas de la materia

Categorías de Chomsky

Lenguajes regulares

- Modelos que representan lenguajes regulares
 - Gramaticas regulares
 - Automatas finitos
 - AFD
 - Func. de transicion generalizada (2do parametro)
 - AFND
 - Func. de transicion generalizada (2do parametro)
 - 2d. generalizacion de la funcion de transicion (1er parametro)
 - AFND- λ
 - Clausura
 - Funcion de transicion extendida en el 2do parametro
 - 2da extension de δ
 - Expresiones regulares
- Conversiones entre modelos
 - AFND \rightarrow AFD
 - AFND- λ \rightarrow AFND
 - ER \rightarrow AFND- λ
 - AFD \rightarrow ER
 - GR \rightarrow AFND
 - AFD \rightarrow GR

AFDs: Minimizacion & Indistinguibilidad

- Indistinguibilidad de estados
 - Def.
 - Teo asoc #1: si $p \equiv q \Rightarrow \forall \omega, \delta(p, \omega) \equiv \delta(q, \omega)$
 - Teo asoc #2: $La \equiv$ es rel. De equiv.
- Indistinguibilidad de orden k (Def. y 5 propiedades)
- Automata Minimo
 - Def. de aut. Minimo
 - Demostracion de equivalencia con 2 teoremas.
 - Algoritmo de minimizacion
 - Demostracion de que el Mr es realmente minimo
- Propiedades de lenguajes regulares
 - Union
 - Concatenacion
 - Complementacion
 - Interseccion
 - Algebra de Boole
 - Union e Interseccion finita
 - Union infinita
 - Todo lenguaje finito es regular
- Problemas decidibles sobre Lenguajes regulares
 - Pertenencia
 - Finitud
 - Vacuidad
 - Equivalencia
- Lema de pumping para Lenguajes regulares
 - Teorema intermedio
 - Lema de pumping y demostracion
 - Como utilizarlo

Lenguajes Independientes de contexto y Gramaticas GLC

- GLCs
 - Arbol de derivacion
 - Gramaticas ambiguas
 - LIC intrinsecamente ambiguo
- Lema de pumping para Lenguajes independiente de contexto
 - Enunciado y demostracion
 - Como usarlo
 - Probar que $L = \{0^n 1^n 2^n : n \geq 1\}$ no es LIC
- Propiedades sobre lenguajes libres de contexto
 - Union
 - Concatenacion
 - L^+ y L^*
 - Interseccion
 - Complemento
- Probar que $L = \{\omega\omega : \omega \in \Sigma^*\}$ no es LIC

Automatas de pila

- ¿Por qué y para que existen como abstracciones?
- APND

- Def.
- Conf. Inst.
- Relacion entre la CI y la funcion δ
- Lenguaje reconocido por P.V. y por E.F.
- Conversion y Equivalencia:
 - Yendo de un APND-porPilaVacía a un APND-porEstadoFinal
 - Yendo de un APND-porEstadoFinal a un APND-porPilaVacía
- APDet. – condiciones para su definicion
- Conversiones
 - GLC \rightarrow APND
 - Conversiones: Equivalencia GLC y APs (En los dos sentidos)...(las demostraciones estan en la carpeta de soledad y en el apunte de la cátedra).
- Lenguajes libres de contexto determinísticos
 - Teorema 1: Para todo APD M existe otro APD M' equivalente que siempre consume la cadena de entrada
 - Teorema 2: El conjunto de los lenguajes libres de contexto determinísticos (LICDs) es cerrado con respecto a la complementacion.

Intersección entre Leng. Reg. y LICs

Simplificación de Gramáticas GLC

- Def. Simbolo Alcanzable
- Def. Simbolo Activo
- Def. Gramatica reducida.
- Algoritmo para obtener simbolos alcanzables
- Algoritmo para obtener simbolos activos
- Algoritmo para obtener simbolos anulables
- Algoritmo para obtener una gramática propia
- Def. gramática propia

Parsers (Analizadores Sintácticos)

AS Descendentes

- No predictivos, no determinísticos
- Predictivos (método procedural)
- Predictivos (método con tabla)
- Def. de gramática LL(1), y propiedades
- Modificación de una G dada:
 - Eliminación recursión a izquierda inmediata
 - Eliminación de recursión a izquierda completa
 - Factorización de gramáticas
- LL(k) – Primerosk, siguientesk, SDK

- Def
- Def de δ
- Config Instantánea de una MT
- Transiciones de una MT (entre CIs)
- Notación en forma de grafo y notación en forma de Tabla
- Lenguaje aceptado
- MT multicinta
- MT-No Determinística
- Def.
- Emulando una MTND con una MT multicinta

Analizadores Ascendentes

- Comparación con los descendentes, las operaciones y que hace cada una
- Pivote
- Algoritmo general de AS Ascendente
- Parsers por precedencia
 - Relaciones de precedencia
 - Construcción tabla de precedencia
 - Gramática “ppropia”
 - Gramática de precedencia
 - Gramática unívocamente inversible
 - Gramática de precedencia simple
 - Algoritmo de AS Asc. Usando relaciones de precedencia
 - Seguimiento de una cadena usando relaciones de precedencia
- Parsers LR
 - Defs. Prefijo viable, ítem, ítem válido, ítem completo
 - AFND que reconoce los prefijos viables de una gramática
 - Teoremas derivados de ese AFND (parte confusa...)
 - Algoritmo general de parsers LR
 - Categorías de parser LR
 - LR(0)
 - Características. Ventajas y limitaciones
 - Construcción de la tabla de parseo (y de su previo AFD asociado)
 - Posibles fuentes de conflicto.
 - SLR – Diferencias con LR(0)
 - LR(1)
 - Comparación con LR(0) y SLR. Por qué es más potente. Y por qué es más costoso y complejo.
 - Def. de ítem e ítem válido nuevas.
 - Nueva def. de función de clausura (afecta la cantidad de estados del AFD asociado)
 - Construcción de la tabla de acción LR(1)
 - LALR
 - Def. de ‘núcleo’ de un ítem.
 - Construcción del AFD
 - Posibles conflictos que pueden aparecer con un LALR

Gramáticas de Atributos (GDAs)

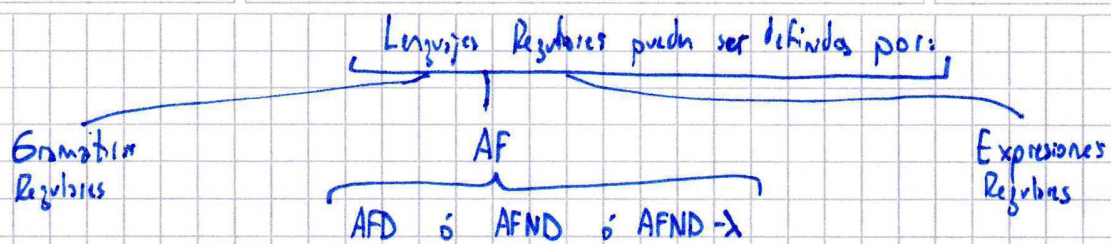
- Defs. Iniciales
- Atributo, regla de valoración, atributo sintetizado, atributo heredado
- Grafo de dependencias (como construirlo)
- Pasos para obtener una valoración
- Ordenamiento topológico
- DDS
- GDA
- L-atribuidas y S-atribuidas
- Esquemas de traducción (se le agregan acciones semánticas y con algunas restricciones)

Compilación y sistemas de tipos

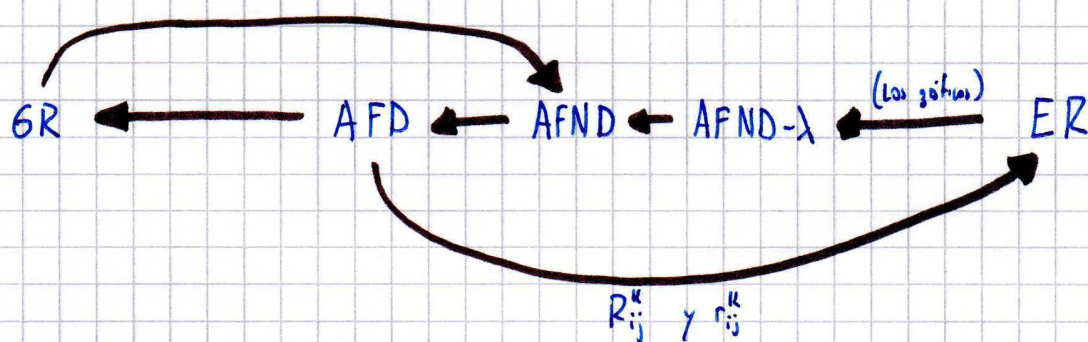
- Pasos de la compilación
- Expresión de tipo, sistema de tipo, sistema de tipo seguro, lenguaje fuertemente tipado
- Equivalencia de expresiones de tipo
- Coerción
- Algoritmo de comprobación estructural de tipos
- Generación de código intermedio
- Código de tres direcciones
- Tipos de operaciones

Maquina de turing

- MT Det.



o por extensión!



Gramática: $\langle V_N, V_T, P, S \rangle$

$V_T = \Sigma$

V_N : Variables o categorías sintácticas

P : \odot de $\alpha \rightarrow \beta$

$S \in V_N$

$$P \subseteq \overbrace{(V_N \cup V_T)^* V_N (V_N \cup V_T)^*}^{\alpha} \times \overbrace{(V_N \cup V_T)^*}^{\beta}$$

FS:

S es FS

Si $\alpha\beta\gamma$ es FS y $(\beta \rightarrow \delta) \in P \Rightarrow \alpha\delta\gamma$ es FS

} Ser FS es un prop. de una cadena de constantes con respecto a una gramática G

Derivación directa: Si $\alpha\beta\gamma \in (V_N \cup V_T)^*$ y $(\beta \rightarrow \delta) \in P$ entonces $\alpha\beta\gamma \xrightarrow{G} \alpha\delta\gamma$ es la deriv. directa. La deriv. directa es una RELACION

\xrightarrow{G}^+ , \xrightarrow{G}^* claus. trans. y claus. trans. reflexiva de \xrightarrow{G} ; \xrightarrow{G}^k es la potencia k

Leng. generado por G :

$$L(G) \text{ o } L(G) = \{ \alpha \in V_T^* : S \xrightarrow{G}^+ \alpha \}$$

Gramáticas regulares (tipo 3): Si todas las produ. son de la forma $A \rightarrow x$ o $A \rightarrow xB$ es lineal a derecha.

Si son de la forma $A \rightarrow x$ o $A \rightarrow Bx$ son lineal a izq.

($A, B \in V_N$ y $x \in V_T^*$)

AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$

Q : \odot finito de estados
 Σ : \odot " de símbolos
 $\delta: Q \times \Sigma \rightarrow Q$
 $q_0 \in Q$
 $F \subseteq Q$

$\hat{\delta}$ func. de trans. generalizada: $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$

$$\hat{\delta}(q, \lambda) = q$$

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a) \quad // (\text{donde } w \in \Sigma^* \text{ y } a \in \Sigma)$$

$$\rightarrow \text{Notar que: } \underline{\hat{\delta}(q, a)} = \delta(\hat{\delta}(q, \lambda), a) = \underline{\delta(q, a)}$$

Cadena aceptada por un AFD M : α es aceptada por M si $\hat{\delta}(q_0, \alpha) = q$
 tal que $q \in F$ (M acepta α sii $\hat{\delta}(q_0, \alpha) \in F$)

Leng. definido por un AFD M :

$$L(M): L(M) = \{ \alpha : \hat{\delta}(q_0, \alpha) \in F \}$$

Notar que para la $\hat{\delta}$ generalizada:

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

AFND: $M = \langle Q, \Sigma, \delta, q_0, F \rangle$

$\delta : Q \times \Sigma \rightarrow P(Q)$ [Esto es lo que cambia!!!]

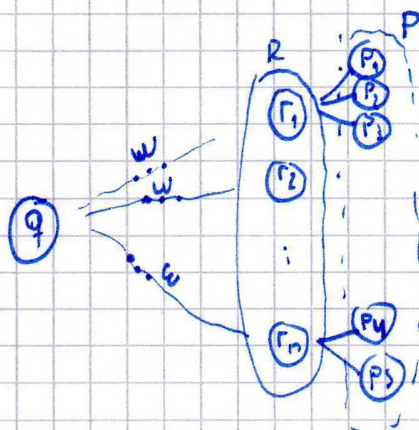
Func. de transición generalizada para AFNDs ($\hat{\delta}$) // generalizada sobre 2º parámetro

$\hat{\delta} : Q \times \Sigma^* \rightarrow P(Q)$

// $P \equiv \mathbb{P}$

$\hat{\delta}(q, \lambda) = \{q\}$

$\hat{\delta}(q, wa) = \{p : [\exists r (r \in \hat{\delta}(q, w) \wedge p \in \delta(r, a))]\}$ [I]



Cadena aceptada por un AFND M: α es aceptada por M sii

$$[\hat{\delta}(q_0, \alpha) \cap F] \neq \emptyset$$

Leng. definido por un AFND M:

$$L(M) = \{ \alpha : (\hat{\delta}(q_0, \alpha) \cap F) \neq \emptyset \}$$

Equiv. entre $\hat{\delta}$ y δ :

$$\hat{\delta}(q, a) = \{p : \exists r (r \in \hat{\delta}(q, \lambda) \wedge p \in \delta(r, a))\}$$

$$= \{p : \exists r (r \in \{q\} \wedge p \in \delta(r, a))\}$$

$$= \{p : p \in \delta(q, a)\} = \delta(q, a)$$

2da extensión de $\hat{\delta}$ (ahora sobre el 1º parámetro):

$$\hat{\hat{\delta}} : P(Q) \times \Sigma^* \rightarrow P(Q)$$

$$\left[\hat{\hat{\delta}}(P, x) = \bigcup_{q \in P} \hat{\delta}(q, x) \right]$$

Ojo! tanto $\hat{\delta}$ como $\hat{\hat{\delta}}$ los puedo notar simplemente como " δ "

De AFND \rightarrow AFD

Recordar que:

$$\hat{\delta} : P(Q) \times \Sigma^* \rightarrow P(Q)$$

$$\hat{\delta}(P, w) = \bigcup_{q \in P} \hat{\delta}(q, w)$$

Construcción / Definición del AFD M' a partir del AFND M

¿Cómo ir de un AFND M a un AFD $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$?

$Q' = \{ [q_0, \dots, q_n] : \{q_0, \dots, q_n\} \in P(Q) \}$ \leadsto Pongo todos los elem de $P(Q)$ como estados de Q' !!

$\Sigma' = \Sigma$

$\delta' / \delta'([q_0, \dots, q_n], a) = [p_0, \dots, p_k]$
 si

$\hat{\delta}(\{q_0, \dots, q_n\}, a) = \{p_0, \dots, p_k\}$

$q'_0 = [q_0]$

$F' = \{ [q_0, \dots, q_n] \in Q' : \{q_0, \dots, q_n\} \cap F \neq \emptyset \}$

¿Cómo demostrar que $L(M') = L(M)$? Tiene 2 partes1º) Demostrar que $\delta'([q_0], w) = [q_0, \dots, q_n]$ si $\delta(q_0, w) = \{q_0, \dots, q_n\}$ Por inducción en $|w|$:

$$|w|=0 \Rightarrow w=\lambda; \quad \delta'([q_0], \lambda) = [q_0] \quad \text{y} \quad \delta(q_0, \lambda) = \{q_0\} \quad (\text{por def. de } \delta \text{ y } \delta')$$

$$\text{Assume que para } |w|=n \text{ vale que: } \delta'([q_0], w) = [q_0, \dots, q_n] \text{ si } \delta(q_0, w) = \{q_0, \dots, q_n\}$$

qpp la cadena wa : $qpqa$:

$$\delta'([q_0], wa) = [q_0, \dots, q_n] \text{ si } \delta(q_0, wa) = \{q_0, \dots, q_n\}$$

(1): prop. de δ

(2): H.I. asumiendo que esto es así.

$$\delta'(q_0, wa) \stackrel{(1)}{=} \delta'(\underbrace{\delta(q_0, w)}_{[p_{i_0}, \dots, p_{i_j}]}, a) = \delta'([p_{i_0}, \dots, p_{i_j}], a)$$

(Por H.I. : $\delta'(q_0, w) = [p_{i_0}, \dots, p_{i_j}]$ sii $\delta(q_0, w) = \{p_{i_0}, \dots, p_{i_j}\}$); luego:

$$\delta(q_0, wa) \stackrel{(2)}{=} \delta(\delta(q_0, w), a) \stackrel{(2)}{=} \delta(\{p_{i_0}, \dots, p_{i_j}\}, a)$$

Por def. de δ' :

$$\delta'([p_{i_0}, \dots, p_{i_j}], a) = [r_{i_0}, \dots, r_{i_n}] \text{ sii}$$

$$\delta(\{p_{i_0}, \dots, p_{i_j}\}, a) = \{r_{i_0}, \dots, r_{i_n}\}$$

Por lo tanto; recopilando lo anterior:

Por def de δ y por H.I.:

$$\delta'(q_0, wa) = \delta'([p_{i_0}, \dots, p_{i_j}], a) \text{ sii}$$

$$\delta(q_0, wa) = \delta(\{p_{i_0}, \dots, p_{i_j}\}, a)$$

Y por def de δ'

$$\delta'([p_{i_0}, \dots, p_{i_j}], a) = [r_{i_0}, \dots, r_{i_n}] \text{ sii}$$

$$\delta(\{p_{i_0}, \dots, p_{i_j}\}, a) = \{r_{i_0}, \dots, r_{i_n}\}$$

Y combinando estas dos cosas tenemos que:

$$\delta'(q_0, wa) = [r_{i_0}, \dots, r_{i_n}] \text{ sii } \delta(q_0, wa) = \{r_{i_0}, \dots, r_{i_n}\} \quad \blacksquare$$

2°) Probar que $L(M) = L(M')$

$$x \in L(M) \text{ sii } [\delta(q_0, x) = \{p_{i_0}, \dots, p_{i_n}\} \text{ y } \{p_{i_0}, \dots, p_{i_n}\} \cap F \neq \emptyset]$$

$$\text{sii } [\delta'(q_0, x) = [p_{i_0}, \dots, p_{i_n}] \text{ y } [p_{i_0}, \dots, p_{i_n}] \in F']$$

$$\text{sii } x \in L(M') \quad \blacksquare$$



Basicamente la idea es que la def. de δ' garantiza la equiv. de a un caracter por vez, y ya así recién puedo la equiv. comparando desde q_0/q_0' para toda una cadena!

AFND- λ : $M = \langle Q, Z, \delta, q_0, F \rangle$

$$\delta : Q \times Z \cup \{\lambda\} \rightarrow P(Q)$$

CL_λ de un estado q

$$CL_\lambda : Q \rightarrow P(Q)$$

~~$\{q \in Q : q \in CL_\lambda(q)\}$~~

$CL_\lambda(q) =$ conj. de estados alcanzables desde q siguiendo solo transiciones λ .
pero $q \in CL_\lambda(q)$!!

CL_λ de un conj. de estados : $CL_\lambda : P(Q) \rightarrow P(Q)$

$$CL_\lambda(P) = \bigcup_{q \in P} CL_\lambda(q)$$

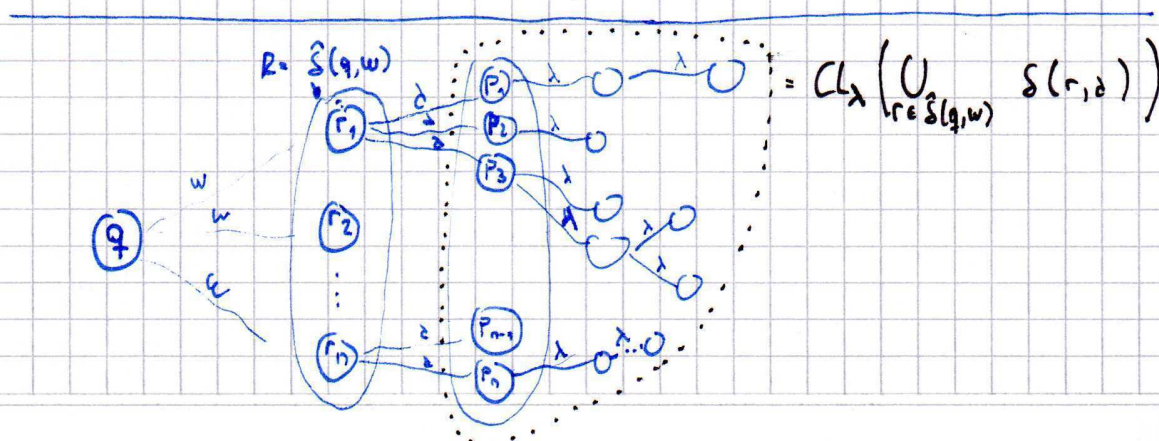
Func. de transición extendida $\hat{\delta}$ (extendida en el Z^* parámetro)

$$\begin{array}{l} \hat{\delta} : Q \times \underbrace{(Z \cup \{\lambda\})^*}_{Z^*} \rightarrow P(Q) \\ \text{Domino} \left[\hat{\delta} : Q \times Z^* \rightarrow P(Q) \right] \end{array}$$

$$\hat{\delta}(q, \lambda) = CL_\lambda(q)$$

$$\hat{\delta}(q, w\alpha) = CL_\lambda(\{p : \exists r (r \in \hat{\delta}(q, w) \wedge p \in \hat{\delta}(r, \alpha))\})$$

$$= CL_\lambda\left(\bigcup_{r \in \hat{\delta}(q, w)} \hat{\delta}(r, \alpha)\right)$$



2ª extensión de \hat{S} (en el 1º parámetro ahora, para aceptar conj. de estados)

$$[\hat{\hat{S}} : \mathcal{P}(Q) \times Z^* \rightarrow \mathcal{P}(Q)]$$

$$\hat{\hat{S}}(P, a) = \bigcup_{p \in P} \hat{S}(p, a)$$

// Nota: que acá uso la "S" no extendida.

$$\hat{\hat{S}}(P, w) = \bigcup_{p \in P} \hat{\hat{S}}(p, w)$$

y ahora que ya tengo definidos esta 2ª extensión; la definición anterior de $\hat{S}(q, wa)$ puedo escribirla:

$$\hat{S}(q, wa) = Cl_\lambda(\hat{\hat{S}}(\hat{S}(q, w), a))$$

Ojo, no hay equivalencia entre $\hat{S}(q, a)$ y $\hat{S}(q, a)$ en el caso de los AFND- λ

$$\begin{aligned} \hat{S}(q, a) &= Cl_\lambda(\hat{\hat{S}}(\hat{S}(q, \lambda), a)) \\ &= Cl_\lambda(\hat{\hat{S}}(Cl_\lambda(q), a)) \neq \hat{S}(q, a) \end{aligned}$$

Cadena aceptada por un AFND- λ M ; y Leng. definido

α es aceptado por M sii $\hat{S}(q_0, \alpha) \cap F \neq \emptyset$

$$\mathcal{L}(M) = \{ \alpha : \hat{S}(q_0, \alpha) \cap F \neq \emptyset \}$$

AFND- $\lambda \rightarrow$ AFND (1)

AFND- λ \rightarrow AFND
M M'

Def / Construcción :

Sea M un AFND- $\lambda : \langle Q, \Sigma, \delta, q_0, F \rangle$

Defino M' un AFND tq:

$$q_0' = q_0 ; \Sigma' = \Sigma ; Q' = Q$$

$$F' = \begin{cases} F \cup \{q_0\} & \text{si } C_{\lambda}(q_0) \cap F \neq \emptyset \\ F & \text{otherwise} \end{cases}$$

$$\delta'(q, a) = \hat{\delta}(q, a)$$

Probar que $L(M') = L(M)$

1º) Probar que $\boxed{\delta'(q_0, w) = \hat{\delta}(q_0, w) \quad \forall w : |w| \geq 1}$

Inducción en $|w|$

Caso base: $|w| = 1 \Rightarrow w = a$; y por def. de δ^*

$$\delta'(q_0, a) = \hat{\delta}(q_0, a)$$

Assume válido por $|w| = n$; qpp $|w| > n$ (para una cadena $x = wa$)

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a) \stackrel{x \text{ HI}}{=} \delta'(\hat{\delta}(q_0, w), a) \quad \checkmark (1)$$

Veremos que $\forall P \subseteq Q, \delta'(P, a) = \hat{\delta}(P, a)$ pues:

$$\delta'(P, a) = \bigcup_{q \in P} \delta'(q, a) = \bigcup_{x \text{ HI}} \bigcup_{q \in P} \hat{\delta}(q, a) = \hat{\delta}(P, a) \quad \checkmark (2)$$

Por lo tanto, si hago $P = \hat{\delta}(q_0, w)$ me queda

$$\delta'(q_0, wa) = \delta'(\hat{\delta}(q_0, w), a) = \delta'(P, a) = \hat{\delta}(P, a) = \hat{\delta}(\hat{\delta}(q_0, w), a) = \hat{\delta}(q_0, wa) \quad \blacksquare$$

AFND $\rightarrow \lambda \rightarrow$ AFND (2)

2º) Ahora sí, probar que $L(M) = L(M')$; o sea $\forall \alpha \in \Sigma^*$, $\alpha \in L(M)$ sii $\alpha \in L(M')$

2.1) Para $\alpha = \lambda$ ($|\alpha| = 0$)

$\alpha \in L(M)$ sii $\delta(q_0, \lambda) \cap F \neq \emptyset$

sii $\delta_\lambda(q_0) \cap F \neq \emptyset$

sii $F' = F \cup \{q_0\}$

sii $q_0' \in F'$ o sea, $\lambda \in L(M')$

o sea:

$\lambda \in L(M)$ sii $\lambda \in L(M')$

2.2) Para $\alpha \neq \lambda$, $|\alpha| \geq 1$

$\alpha \in L(M)$ sii $\delta(q_0, \alpha) \cap F \neq \emptyset$

sii $\delta'(q_0, \alpha) \cap F' \neq \emptyset$

sii $\alpha \in L(M')$

Así habrá que partir a los 2 casos para F' :)

Expresiones Regulares

\emptyset denota al leng. / conj. ~~vacío~~ \emptyset

λ denota $\{\lambda\}$

Para cada $a \in \Sigma$, a denota $\{a\}$

y siguiendo: Si 'r' y 's' denota los lenguajes 'R' y 'S', entonces

$r|s$ denota $R \cup S$

rs denota RS

r^* denota R^*

r^+ denota R^+

} donde dado un leng L , :

$$L^0 = \{\lambda\}$$

$$L^1 = L$$

$$L^n = LL^{n-1}$$

$$L^* = \bigcup_{n \geq 0} L^n$$

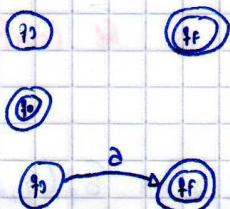
$$L^+ = \bigcup_{n \geq 0} L^n, \quad L^+ = LL^*, \quad L^* = L^+ \cup \{\lambda\}$$

De ER \rightarrow AFND- λ

Con un único est. final y sin transiciones a partir de él

Casos Base:

- \emptyset
- λ
- a



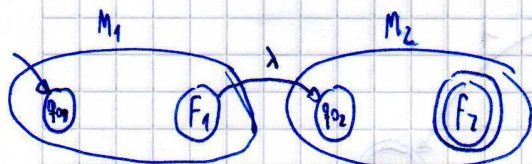
Casos Recurativos

(Inducción a la cont. de operaciones de la E.R.)

Base de las dems. siguientes!

$r_1 r_2$

M:



Por HI: $\exists M_1, M_2$ tq $L(M_1) = L(r_1)$ y $L(M_2) = L(r_2)$

$M_1 = \langle Q_1, \Sigma_1, \delta_1, q_{01}, F_1 \rangle$; $F_1 = \{f_1\}$ (único est. final!)

$M_2 = \langle Q_2, \Sigma_2, \delta_2, q_{02}, F_2 \rangle$; $F_2 = \{f_2\}$ (único est. final!)

$M = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_{01}, \{f_2\} \rangle$

donde:

$$\delta \begin{cases} \delta(f_1, \lambda) = \{q_{02}\} \\ \forall q \in (Q_1 - \{f_1\}), \forall a \in \Sigma_1 \cup \{\lambda\} \\ \forall q \in Q_2, \forall a \in \Sigma_2 \cup \{\lambda\} \end{cases}$$

$$\delta(q, a) = \delta_1(q, a)$$

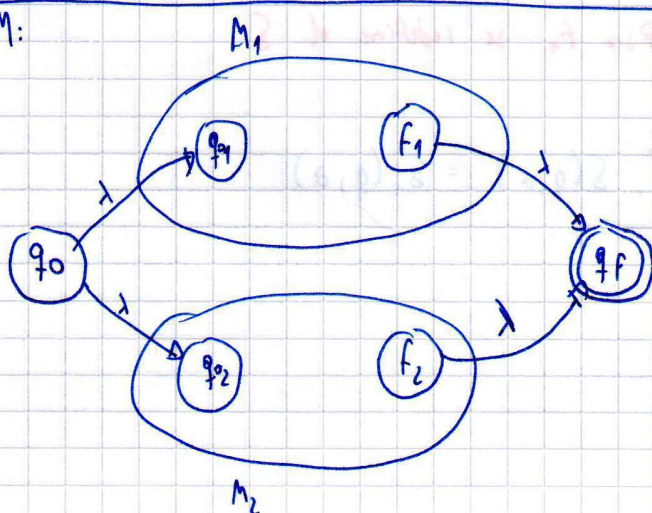
$$\delta(q, a) = \delta_2(q, a)$$

Sólo a "f1" por le doy int. especial

No lo voy a copiar de nuevo para los otros casos recursivos...

r_1 / r_2

M:



$$M: \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{q_f\} \rangle$$

$$\delta(q_0, \lambda) = \{q_{01}, q_{02}\}$$

$$\delta(f_1, \lambda) = \delta(f_2, \lambda) = \{q_f\} \quad \text{por } f_1 \text{ y } f_2 \text{ se redefinen los "}\delta()\text{"}$$

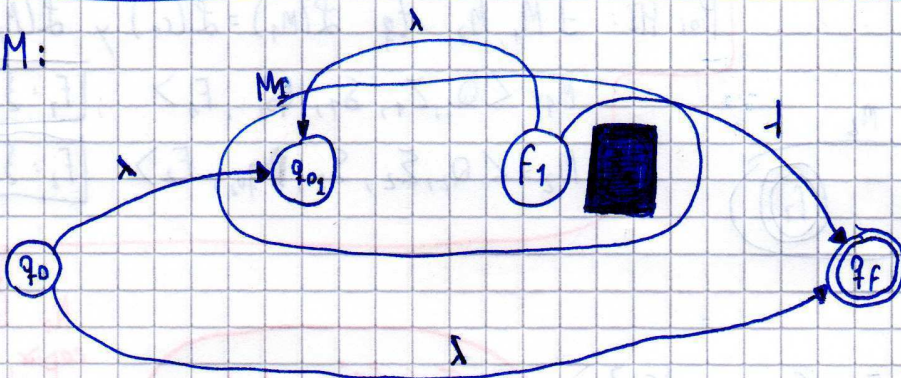
$$\forall a \in \Sigma_1 \cup \{\lambda\}, q \in Q_1 - \{f_1\}, \quad \delta(q, a) = \delta_1(q, a)$$

$$\forall a \in \Sigma_2 \cup \{\lambda\}, q \in Q_2 - \{f_2\}, \quad \delta(q, a) = \delta_2(q, a)$$

los otros símbolos y estados van los $\delta()$ de M_1 y M_2

r_1^*

$M:$



$$M_1: \langle Q_1, \Sigma_1, \delta_1, q_{01}, \{f_1\} \rangle$$

$$M: \langle \{q_0, q_f\} \cup Q_2, \Sigma_1 \cup \{\lambda\}, \delta, q_0, \{q_f\} \rangle$$

$$\delta(q_0, \lambda) = \{q_{02}, q_f\}$$

$$\delta(f_1, \lambda) = \{q_{02}, q_f\} \quad \text{Por } f_1 \text{ se redefine el } \delta$$

$$\forall a \in \Sigma_1 \cup \{\lambda\}, q \in Q_1 - \{f_1\}, \quad \delta(q, a) = \delta_1(q, a)$$

AFD \rightarrow ER (1)

- Defino que es R_{ij}^k como abstracción, en relación a un AFD M

- R_{ij}^k recursivamente $\begin{matrix} \nwarrow R_{ij}^k \\ \searrow R_{ij}^0 \end{matrix}$; $k \geq 1$

Definimos que es R_{ij}^k :

Vamos a ver:

R_{ij}^k : Caminos que van del estado q_i al q_j pasando por estados cuyo índice es a lo sumo k . R_{ij}^k es un conj. de cadenas! (o sea... es un lenguaje)

Def rec:

$$\text{Para } k \geq 1 : R_{ij}^k = \underbrace{R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}}_{\text{Caminos que van del estado } q_i \text{ a } q_j \text{ pasando por } q_k} \cup \underbrace{R_{ij}^{k-1}}_{\text{Caminos que van de } q_i \text{ a } q_j \text{ sin pasar por } q_k}$$

$$\text{Para } k=0 : R_{ij}^0 = \begin{cases} \{a : \delta(q_i, a) = q_j\} \cup \{\lambda\} & \text{si } i=j \\ \{a : \delta(q_i, a) = q_j\} & \text{si } i \neq j \end{cases}$$

Construcción de una expresión regular 'r' tq $\mathcal{L}(r) = R_{ij}^k$, $\forall i, j, k$ posibles en un AFD dado:

Caso base, para $k=0$. Construir una ER tq $\mathcal{L}(r) = R_{ij}^0$. Llamari r_{ij}^0 a esa ER

$$r_{ij}^0 = \begin{cases} a_1 \dots a_p & \text{si } \delta(q_i, a_h) = q_j \text{ y } q_i \neq q_j \\ a_1 \dots a_p / \lambda & \text{si } [\quad] \text{ y } q_i = q_j \\ \emptyset & \text{si } \nexists a : \delta(q_i, a) = q_j \text{ y } q_i \neq q_j \\ \lambda & \text{si } [\nexists a : \delta(q_i, a) = q_j] \text{ y } q_i = q_j \end{cases} \quad \left. \begin{array}{l} \text{Es trivial ver que} \\ \mathcal{L}(r_{ij}^0) \text{ definido así, es} \\ \text{igual a } R_{ij}^0 \end{array} \right\}$$

Paso de definición de r_{ij}^k inductivo: ~~asumo~~ que tengo definida satisfactoriamente a una ER r_{ij}^h , \forall

$\forall i, j$ y $\forall h < k$; tq $\mathcal{L}(r_{ij}^h) = R_{ij}^h$. Quiero construir una ER r_{ij}^k asociada a R_{ij}^k .

Propongo $r_{ij}^k = r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} \mid r_{ij}^{k-1}$ // por HI tengo definida r_{**}^{k-1}

$$\begin{aligned} \text{Opera : } \mathcal{L}(r_{ij}^k) &= \mathcal{L}(r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} \mid r_{ij}^{k-1}) = \mathcal{L}(r_{ik}^{k-1}) \mathcal{L}(r_{kk}^{k-1})^* \mathcal{L}(r_{kj}^{k-1}) \cup \mathcal{L}(r_{ij}^{k-1}) \\ &= \underbrace{R_{ik}^{k-1}}_{\text{por HI}} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup \underbrace{R_{ij}^{k-1}}_{\text{por def } R_{ij}^h} = R_{ij}^k \end{aligned}$$

~~AFD~~ AFD \rightarrow ER (1)

Listo, ya s e:

- Como definir R_{ij}^k recursivamente
- Como definir r_{ij}^k recursivamente, uno a uno a cada R_{ij}^k

Ahora queda ver qu  es la ER que denota a $\mathcal{L}(M)$

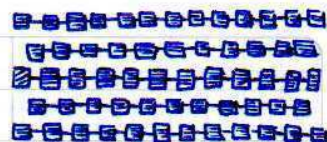
$$\mathcal{L}(M) = \bigcup_{q_f \in F} R_{q_1, q_f}^n \quad \text{donde } \begin{array}{l} q_1 = \text{estado inicial} \\ n = \text{cant. de estados en el AFD } M \end{array}$$

$$= \bigcup_{q_f \in F} \mathcal{L}(r_{q_1, q_f}^n) = \mathcal{L}(r_{q_1, q_{f_1}}^n \mid r_{q_1, q_{f_2}}^n \mid r_{q_1, q_{f_3}}^n \mid \dots \mid r_{q_1, q_{f_n}}^n)$$

y esto es la ER que denota a $\mathcal{L}(M)$ ■

GR \rightarrow AFND

GR \rightarrow AFND

 [lineal \rightarrow derecho]
Grismita es:

$$G = \langle \underbrace{V_N}_{Q}, \underbrace{V_T}_{\Sigma}, \underbrace{P}_{\delta}, \underbrace{S}_{q_0} \rangle$$

} puede decir: "más o menos es..."

Como construir un M a partir de una G

Sea $G = \langle V_N, V_T, P, S \rangle$, lineal \rightarrow derecho $(A \rightarrow x B \text{ o } A \rightarrow x)$

Entonces M se construye así:

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle \quad \text{tq}$$

$$\bullet \Sigma = V_T$$

$$\bullet Q = \{q_f\} \cup \{q_A : A \in V_N\}$$

$\forall A \in V_N, \exists q_A \in Q$

// de q₀ a q_f + estados finales q_f / pero puede haber + estados finales

$$\bullet q_0 \equiv q_s$$

$$\bullet F = \left[\begin{array}{l} q_f \in F \\ \forall (A \rightarrow x B) \in P \Rightarrow q_A \in F \end{array} \right]$$

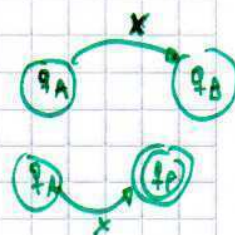


Def de F

$$\bullet \delta \text{ se define así: } \delta: Q \times \Sigma \rightarrow P(Q)$$

$$\text{a) } q_B \in \delta(q_A, x) \quad \text{sii} \quad A \rightarrow x B \in P$$

$$\text{b) } q_f \in \delta(q_A, x) \quad \text{sii} \quad A \rightarrow x \in P$$

Def de δ

AFD \rightarrow GRSea AFD $M = \langle Q, \Sigma, \delta, q_0, P \rangle$ $G = \langle V_N, V_T, P, S \rangle$ $V_N = Q$ // A_p es el no-terminal asociado al estado p en Q $V_T = \Sigma$ $S = A_{q_0}$

$P : \left[\begin{array}{llll} A_p \rightarrow a A_q & \in P & \text{sii} & \delta(p, a) = q \\ A_p \rightarrow a & \in P & \text{sii} & \delta(p, a) = q \text{ y } q \in F \\ S \rightarrow \lambda & \in P & \text{sii} & q_0 \in F \end{array} \right.$

Construcción de 6

Indistinguibilidad (def)

- Teo asoc. #1: $p \equiv q$ indist. $\Rightarrow \forall \alpha, \overline{\delta(p, \alpha)} = \overline{\delta(q, \alpha)}$
- Teo asoc. #2: La " \equiv " es una rel. de equiv.

Indistinguibilidad de orden K (\equiv^K)

- 5 propiedades

AFD Mínimo

- Def de M'
- Visto que M y M' son equiv. 2 teoremas: $\left[\delta(q, \alpha) = r \Rightarrow \delta'(q, \alpha) = [r] \right]$
 $\left[\delta'([q], \alpha) = [r] \Rightarrow \exists s \in Q: \delta(q, \alpha) = s \wedge s \equiv r \right]$
- Alg. de Minimización
- Teo: ningún aut. que reconozca el mismo lenguaje de un aut. mínimo, tiene menos estados
- Teo: Sean M y M' . Si todo par de palabras que conducen a est. difs. a M , tb. conducen a ests. diferentes a $M' \Rightarrow \#Q \leq \#Q'$

Estados indistinguibles:

Dado $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ AFD; p y q son est. indistinguibles ($p \equiv q$) definidos así:

$$(p \equiv q) \text{ sii } \forall \alpha, [\delta(p, \alpha) \in F \text{ sii } \delta(q, \alpha) \in F]$$

Teo asociado: si p, q son indist. entonces al seguir cualq. cadena α desde ambos estados, los estados p' y q' a los que se llegue tb. serán indistinguibles.

Dem: (por el absurdo). Sup. que el teo no vale. O sea, suponemos que: $\exists \alpha$ tq:

$$p \equiv q; \delta(p, \alpha) = p'; \delta(q, \alpha) = q'; p' \not\equiv q'$$

$$p' \not\equiv q' \Rightarrow \exists \beta \text{ tq } \delta(p', \beta) \in F \text{ y } \delta(q', \beta) \notin F \text{ (o al revés, no importa)}$$

$$\begin{aligned} \text{Pero entonces: } & \delta(p', \beta) = \delta(\overline{\delta(p, \alpha)}, \beta) = \delta(p, \alpha\beta) \in F \\ \text{y} & \delta(q', \beta) = \delta(\delta(q, \alpha), \beta) = \delta(q, \alpha\beta) \notin F \end{aligned} \left. \begin{array}{l} \text{pero } p \equiv q, \\ \text{o sea, llegué} \\ \text{a un ABSURDO} \end{array} \right\}$$

Teo: \equiv es una rel. de equivalencia

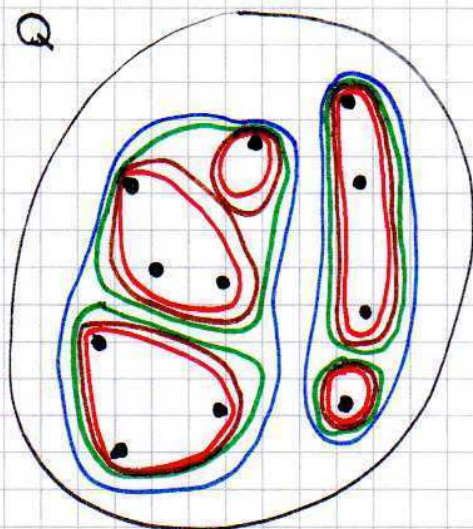
- es trivial demostrar esto
- reflexiva: $\forall p, p \equiv p$ (trivial)
 - simétrica: $\forall p, q (p \equiv q \Rightarrow q \equiv p)$
 $p \equiv q \Rightarrow [\forall \alpha \in \Sigma^*, \delta(p, \alpha) \in F \text{ sii } \delta(q, \alpha) \in F]$
 $\Rightarrow q \equiv p$
 - transitiva: Si $p \equiv q \wedge q \equiv r \Rightarrow p \equiv r$. Veámoslo por el absurdo:
 Si $\exists \beta / \delta(p, \beta) \in F \wedge \delta(r, \beta) \notin F$ (o al revés, no importa)
 \Rightarrow Como $\delta(p, \beta) \in F$ y $p \equiv q \Rightarrow \delta(q, \beta) \in F$
 y como $\delta(q, \beta) \in F$ y $q \equiv r \Rightarrow \delta(r, \beta) \in F$... pero habíamos supuesto que $\delta(r, \beta) \notin F \rightarrow$ ABSURDO! ■

Indisting. de orden K (\equiv^K).

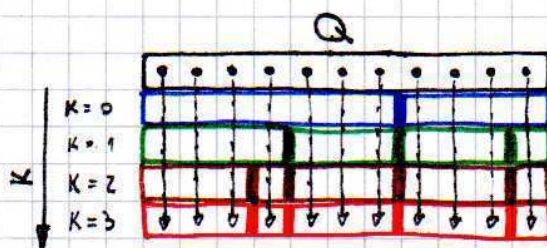
$$p \equiv^K q \text{ si: } \forall \alpha \in \Sigma^*, (|\alpha| \leq K) \Rightarrow (\delta(p, \alpha) \in F \text{ sii } \delta(q, \alpha) \in F)$$

Props. de \equiv^K :

- Sólo por ABS
- \equiv^K es una relación de equivalencia
 - $\equiv^{K+1} \subseteq \equiv^K$ // o sea: si $p \equiv^{K+1} q \Rightarrow p \equiv^K q$... tiene sentido
 - $(Q / \equiv^K) = \{Q-F, F\}$ si $Q-F \neq \emptyset \wedge F \neq \emptyset$
 - // (Q / \equiv^K) significa separar
 - // Q en clases de equivalencia
 - // creadas por la relación \equiv^K
 - $p \equiv^{K+1} q$ sii $\forall \alpha \in \Sigma, \delta(p, \alpha) \equiv^K \delta(q, \alpha)$
- Por ABSURDO
- Por inducción en "n"
- NOTA
- Se usa como criterio de parada en el alg. de minimización.
- // Una vez que ya no puede seguir dividiendo los c.d.e. en más chicas, ya no lo voy a poder hacer.

Ejemplo:

Q/\equiv_0
 Q/\equiv_1
 Q/\equiv_2
 Q/\equiv_3



Notar que: $Q/\equiv_0 \neq Q/\equiv_1$
 $Q/\equiv_1 \neq Q/\equiv_2$
 $Q/\equiv_2 = Q/\equiv_3$

Otra manera de verlo. Los elementos
 se van ubicando en particiones
 cada vez + chicas en cada iteración.

y así puedo parar de dividir... pq si
 son \equiv hasta ahí, lo van a ser siempre.

Demstraciones de las propiedades

① (Trivial — sale por el absurdo)

② $\equiv^{K+1} \subseteq \equiv^K$ (sale por el absurdo)

Sup. que $p \equiv^{K+1} q$ y $p \not\equiv^K q$.

Como $p \equiv^{K+1} q \Rightarrow \forall \alpha \in \Sigma^*, (|\alpha| \leq K+1) \Rightarrow \delta(p, \alpha) \in F$ si $\delta(q, \alpha) \in F$

Y como $p \not\equiv^K q \Rightarrow \exists \beta \in \Sigma^*, (|\beta| \leq K) \wedge \delta(p, \beta) \in F \wedge \delta(q, \beta) \notin F$

(o al revés)

Pero como $|\beta| \leq K+1 \dots$ y $p \equiv^{K+1} q \leadsto$ ABSURDO! ■

Si: $|\beta| \leq K$ entonces
 en particular $|\beta| \leq K+1$

→ Lo que esta propiedad ($\equiv^{K+1} \subseteq \equiv^K$) asegura es que al ir dividiendo los
 estados en particiones, una partición de más bajo nivel (\equiv^{K+1}) nunca
 cruza los bordes de la partición de más alto nivel que la contiene.

$$(3) (Q / \equiv) = \{Q - F, F\} \quad \text{s.t. } (Q - F) \neq \emptyset \wedge F \neq \emptyset$$

Dem: sste de la def. de \equiv^k con $k=0$

$$p \equiv^0 q \text{ s.t. } \forall \alpha \in \mathbb{Z}^* \underbrace{(|\alpha| \leq 0)}_{\alpha = \lambda} \Rightarrow \delta(p, \lambda) \in F \text{ s.t. } \delta(q, \lambda) \in F$$

$$\text{Y como } \delta(r, \lambda) = r, \forall r \in Q; \Rightarrow \delta(p, \lambda) \in F \text{ s.t. } p \in F$$

$$\delta(p, \lambda) \notin F \text{ s.t. } p \notin F \quad \blacksquare$$

$$(4) p \equiv^{k+1} q \text{ s.t. } \forall \alpha \in \mathbb{Z}, \delta(p, \alpha) \equiv^k \delta(q, \alpha)$$

$$\Rightarrow p \equiv^k q \Rightarrow \forall \alpha \in \mathbb{Z}, \delta(p, \alpha) \equiv^k \delta(q, \alpha)$$

Por el absurdo: Sup. $\exists \alpha / \delta(p, \alpha) \not\equiv^k \delta(q, \alpha) \Rightarrow \exists \alpha, |\alpha| \leq k$ y

$$\delta(\delta(p, \alpha), \alpha) \in F \wedge \delta(\delta(q, \alpha), \alpha) \notin F \Leftrightarrow$$

$$\delta(p, \underbrace{\alpha \alpha}_{\leq k+1}) \in F \wedge \delta(q, \underbrace{\alpha \alpha}_{\leq k+1}) \notin F \Rightarrow p \not\equiv^{k+1} q \rightarrow \text{ABSURDO.}$$

~~Por el absurdo: Sup. $\exists \alpha / \delta(p, \alpha) \not\equiv^k \delta(q, \alpha) \Rightarrow \exists \alpha, |\alpha| \leq k$ y~~

\Leftarrow Por el absurdo:

Sup. que $\forall \alpha \in \mathbb{Z}, \delta(p, \alpha) \equiv^k \delta(q, \alpha)$ pero que $p \not\equiv^{k+1} q$

O sea, $\exists \alpha = \alpha', |\alpha| \leq k+1$, t.q.

$$\delta(p, \alpha \alpha') \in F \wedge \delta(q, \alpha \alpha') \notin F$$

O sea:

$$\delta(\delta(p, \alpha), \alpha') \in F \wedge \delta(\delta(q, \alpha), \alpha') \notin F$$

Pero $|\alpha'| \leq k$, y $\delta(p, \alpha) \equiv^k \delta(q, \alpha)$, por lo tanto \rightarrow ABSURDO!

$$⑤ \left(\overset{k+1}{\equiv} = \overset{k}{\equiv} \right) \Rightarrow \forall n \geq 0 \quad \left(\overset{k+n}{\equiv} = \overset{k}{\equiv} \right)$$

Por ind. en 'n'.

Caso BASE: $n=0 \quad \left(\overset{k+0}{\equiv} = \overset{k}{\equiv} \right)$

H.I.:

Sup. que es válido para n ; y qpp $n+1$. O sea, ~~qpp~~ qpp:

$$\left(\overset{k+1}{\equiv} = \overset{k}{\equiv} \right) \Rightarrow \left(\overset{k+n+1}{\equiv} = \overset{k}{\equiv} \right)$$

"antecedente"

Esto puede escribirse como:

$$\forall p, q \in \mathbb{Q} \left(p \overset{k+n+1}{\equiv} q \Leftrightarrow p \overset{k}{\equiv} q \right)$$

O sea que lo que quiero probar es: (lo reescribo de otra manera):

$$\left(\overset{k+1}{\equiv} = \overset{k}{\equiv} \right) \Rightarrow \forall p, q \in \mathbb{Q} \left(p \overset{k+n+1}{\equiv} q \Leftrightarrow p \overset{k}{\equiv} q \right)$$

Lo cual puede probarse usando 3 cosas: (la H.I., el antecedente $\left(\overset{k+1}{\equiv} = \overset{k}{\equiv} \right)$ y la prop. anterior, la ④); de la sig. manera:

$$p \overset{k+n+1}{\equiv} q \stackrel{x \text{ ④}}{\Leftrightarrow} \forall a \in \mathbb{Z}, \delta(p, a) \overset{k+n}{\equiv} \delta(q, a)$$

$$\stackrel{x \text{ HI}}{\Leftrightarrow} \forall a \in \mathbb{Z}, \delta(p, a) \overset{k}{\equiv} \delta(q, a)$$

$$\stackrel{x \text{ ④ y H.I.}}{\Leftrightarrow} p \overset{k+1}{\equiv} q \Leftrightarrow p \overset{k}{\equiv} q$$

↑
por el antecedente que dice que $\left(\overset{k+1}{\equiv} = \overset{k}{\equiv} \right)$

// Idea de la demostración: a el paso inductivo reescribo el problema para llevarlo a que quiero probar que $p \overset{k+n+1}{\equiv} q \Leftrightarrow p \overset{k}{\equiv} q$. Y ahí hago:

$$p \overset{k+n+1}{\equiv} q \stackrel{\text{④}}{\Leftrightarrow} \forall a \delta(p, a) \overset{k+n}{\equiv} \delta(q, a) \stackrel{\text{HI}}{\Leftrightarrow} \forall a \delta(p, a) \overset{k}{\equiv} \delta(q, a) \stackrel{\text{④}}{\Leftrightarrow} p \overset{k+1}{\equiv} q \stackrel{\text{Antecedente}}{\Leftrightarrow} p \overset{k}{\equiv} q$$

¡Bellísimo!

Def. AFD Mínimo : Dado un $M = \langle Q, Z, \delta, q_0, F \rangle$, \exists v. $M' = \langle Q', Z, \delta', q_0', F' \rangle$ equivalente a M , definido como:

- $Q' = (Q / \equiv)$ // los c.d.e. de Q , partidos hasta lo más posible
(hasta que ya no se pueda seguir dividiendo los c.d.e. en más chicas).
- $\delta'([q], a) = [\delta(q, a)]$
- $q_0' = [q_0]$
- $F' = \{ [q] \in Q' : q \in F \}$

¿Cómo ver si: $L(M) = L(M')$?

Dos teoremas:

Teo: $\delta(q, \alpha) = r \Rightarrow \delta'([q], \alpha) = [r]$

→ (Si una transición dada existe en M , entonces en M' existe una transición equivalente entre los c.d.e. de esos mismos estados)

Dem: por inducción en $|\alpha|$

BASE: $|\alpha| = 0 \Rightarrow \alpha = \lambda$.

Sup. válido para $|\alpha| = n$; qpp $|\alpha| = n+1 \Rightarrow \alpha = a\alpha'$.] HI

Qpq: $\delta(q, a\alpha') = r \Rightarrow \delta'([q], a\alpha') = [r]$

Veamos:

$$\delta(q, a\alpha') = r \Rightarrow \delta(\delta(q, a), \alpha') = r \Rightarrow \delta'([\delta(q, a)], \alpha') = [r]$$

$$\text{Pero } \delta'([\delta(q, a)], \alpha') = \delta'(\delta'([q], a), \alpha') \stackrel{HI}{=} \delta'([q], a\alpha')$$

entonces, $\delta'([q], a\alpha') = [r]$ ■ // llegué del antecedente al consecuente.

Teo: $S'([q_0], \alpha) = [r] \Rightarrow \exists s \in Q : \delta(q_0, \alpha) = s \wedge s \equiv r$

(Es lo vuelto del teo anterior... ☺)

Dem: inducción a $|\alpha|$

C-BASE: $|\alpha| = 0 \Rightarrow \alpha = \lambda$. $S'(q_0, \lambda) = q_0$ y $\delta(q_0, \lambda) = q_0 \dots$ vale!

HI: Sup. q' vale para $|\alpha| = n$.

PI: Qpp $|\alpha| = n+1 \Rightarrow \alpha = w\alpha$

$$S'([q_0], w\alpha) = [r] \Rightarrow \exists p \ S'([q_0], w) = [p] \wedge S'([p], \alpha) = [r]$$

$$\Rightarrow \exists q : \delta(q_0, w) = q \wedge q \equiv p \wedge S'([p], \alpha) = [r]$$

$$\Rightarrow \delta(p, \alpha) = s \quad // \text{ por def de } S'$$

donde $s \equiv r$, o sea $s \equiv r$

$$\Rightarrow \exists q : \delta(q_0, w) = q \wedge q \equiv p \wedge \delta(p, \alpha) = s$$

Como $q \equiv p$

$$\Rightarrow \exists q : \delta(q_0, w) = q \wedge \delta(q, \alpha) = s$$

Pero esto, definitivamente quiere decir que:

$$\Rightarrow \delta(q_0, w\alpha) = s \quad ; \quad \text{donde } s \equiv r \quad \blacksquare$$

acá si quisiera
podría poner en lq.
otro estado equivalente
a "p"

Ahora estamos listos para probar que $\mathcal{L}(M) = \mathcal{L}(M')$. (ahora es trivial!)

$$\bullet x \in \mathcal{L}(M) \Rightarrow \delta(q_0, x) = r, r \in F \Rightarrow S'([q_0], x) = [r], [r] \in F' \Rightarrow x \in \mathcal{L}(M')$$

$$\bullet x \in \mathcal{L}(M') \Rightarrow S'([q_0], x) = [r], [r] \in F'$$

$$\Rightarrow \exists s \in Q : \delta(q_0, x) = s \wedge s \equiv r \Rightarrow s \in F \Rightarrow x \in \mathcal{L}(M)$$

def. de F'

Algoritmo de Minimización

 $P \leftarrow \{Q \setminus F, F\}$

// P y P' son "particiones": conjuntos de conjuntos de
// estados. O sea, conj. de ~~conj.~~ "clases de equivalencia"

Repetir

 ~~$P' \leftarrow \emptyset$~~

Para cada $x \in P$ // x es una c.d.e.: un conj. de estados. considerados equivalentes
// en la iteración anterior

Mientras $(\exists e \in x \wedge \neg \text{marcado}(e, x))$ // e es un estado

 $X_1 \leftarrow \{e\}$

// X_1 = conj. de estados. la c.d.e. de todos los ests.
equiv. a " e "

 $\text{marcar}(e, x)$

 para cada $e' \in X$

// voy a buscar cuales otros " e' " son equivs a " e "

si $\neg \text{marcado}(e', x) \wedge \forall a \in \Sigma [\delta(e, a)] \equiv [\delta(e', a)]$

 $X_1 \leftarrow X_1 \cup \{e'\}$

// si $e' \equiv e$, le agrego " e' " a la c.d.e. de " e "

 $\text{marcar}(e', x)$

Fin si
Fin para

 $P' \leftarrow P' \cup \{X_1\}$

// Como P' es un conj. de c.d.e.s, le agrego una
// c.d.e. más, la c.d.e. de " e "

Fin mientras

Fin para

 Si: $P \neq P'$
 $\text{stop} \leftarrow \text{falso}$
 $P \leftarrow P'$

sino

 $\text{stop} \leftarrow \text{verdadero}$

Fin si

hasta stop

Teorema: Sean M y M' un par de AFD's, en donde $M = \langle Q, \Sigma, \delta, q_0, F \rangle$,

$M' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ y M no posee estados inaccesibles. Si todo par de cadenas que conducen a estados diferentes en M conducen también a estados diferentes en M' , entonces $\#Q' \geq \#Q$. Es decir,

$$(\forall \alpha, \beta \in \Sigma^*, \delta(q_0, \alpha) \neq \delta(q_0, \beta) \Rightarrow \delta'(q'_0, \alpha) \neq \delta'(q'_0, \beta)) \Rightarrow |Q| \leq |Q'|.$$

Dem: Se trata de encontrar una función inyectiva de Q en Q' : $Q \rightarrow Q'$.

Recordemos que una función f es inyectiva $x \neq y \Rightarrow f(x) \neq f(y)$. El hecho de haber una función inyectiva de Q en Q' directamente implicará que $|Q| \leq |Q'|$.

- Consideremos la función $g: Q \rightarrow \Sigma^*$ definida por:

$$g(q) = \min \{ \alpha \in \Sigma^* : \delta(q_0, \alpha) = q \}$$

en donde se supone para el criterio mínimo una relación de orden en Σ^* dada por la longitud y el orden lexicográfico.

- Puede definirse entonces una función $f: Q \rightarrow Q'$:

$$f(q) = \delta'(q'_0, g(q))$$

(Alimentar a M' desde su estado inicial hasta $g(q)$).

- Como cualquier par de estados diferentes $p, q \in Q$ vale que

$$\delta(q_0, g(p)) \neq \delta(q_0, g(q)), \text{ por la def. de } g,$$

- entonces también vale ~~por hipótesis~~ ^{que} ~~llevan a estados diferentes en M' también:~~

$$\delta'(q'_0, g(p)) \neq \delta'(q'_0, g(q)) \quad ; \text{ y entonces por la def. de } f():$$

- o sea, todo lleva a decir que $f(p) \neq f(q)$ con lo cual, f resulta inyectiva, y entonces implica que $|Q| \leq |Q'|$.

idea para dem: exponer como cadenas α y $\beta \in g(p), g(q)$

La idea de la demostración es:

a) Defino $g(q): Q \rightarrow \Sigma^*$; asumiendo orden total entre los elementos de Σ^*

b) " $f(q): Q \rightarrow Q'$; esta es la función que voy a probar que es inyectiva

c) Dados 2 estados diferentes p y q , $\left[\begin{array}{l} \text{las cadenas } g(p) \text{ y } g(q) \text{ llevan a est. difs.} \\ f(p) \neq f(q) \end{array} \right] \Rightarrow f() \text{ es inyectiva}$

así como una relación de orden TOTAL entre cadenas, lo cual me permite elegir UNA en particular

las cadenas $g(p)$ y $g(q)$ llevan a estados diferentes en M ; por lo tanto tb. llevan a estados difs. en M' ; y entonces $f()$ es inyectiva

Lema: Sea M_R el autómata reducido correspondiente a $M = \langle Q, \Sigma, \delta, q_0, F \rangle$.

Entonces, cualquier autómata M' que reconozca el mismo lenguaje, no poseerá menos estados que el autómata reducido, o sea:

$$\forall M' : L(M') = L(M_R) \Rightarrow |Q_R| \leq |Q'|$$

Dem: por el absurdo. Supongamos que $\exists M'$ tal que $|Q'| < |Q_R|$. Según el teorema anterior, entonces, deben existir dos cadenas α, β tales que

$$(\delta_R(q_0, \alpha) \neq \delta_R(q_0, \beta)) \wedge (\delta'(q'_0, \alpha) = \delta'(q'_0, \beta))$$

Usa la contrarrecíproca del teo. anterior; con " M' " en el lugar de M .

pero, como $\delta_R(q_0, \alpha)$ y $\delta_R(q_0, \beta)$ son distinguibles por pertenecer a un autómata reducido, M_R , entonces, $\exists \gamma \in \Sigma^*$ tal que

$$\delta_R(q_0, \alpha\gamma) \in F \wedge \delta_R(q_0, \beta\gamma) \notin F \text{ o viceversa, es decir } \alpha\gamma \in L(M_R) \Leftrightarrow \beta\gamma \notin L(M_R).$$

por otro lado, como $\delta'(q'_0, \alpha) = \delta'(q'_0, \beta)$, es decir que

$$\delta(q_0, \alpha\gamma) \in F \Leftrightarrow \delta(q_0, \beta\gamma) \in F$$

$$\text{con lo cual, } \alpha\gamma \in L(M') \Leftrightarrow \beta\gamma \in L(M')$$

pero entonces estaríamos diciendo que $L(M_R) \neq L(M')$ lo cual contradice la hipótesis.

Es bonito! Nada más hay que recordar que $\delta_R(q_0, \alpha)$ y $\delta_R(q_0, \beta)$ son distinguibles. *inventar* y

☺ Efectivamente, dear sole!

Propiedades de LENGUAJES REGULARES

Cerrado respecto de la Unión

Sea L_1 tq $\mathcal{L}(r_1) = L_1$

Sea L_2 tq $\mathcal{L}(r_2) = L_2$

Sea la expr. regular:

$r_3 = r_1 \mid r_2$ denota a $L_1 \cup L_2$;

$\Rightarrow L_1 \cup L_2$ es regular!

Para cada lng. reg., existe una expr. reg. que lo denota.

Los expr. regulares denotan solo lng. regulares

y r_3 genera solo lng. regulares

Cerrado respecto de la concatenación

(Si L_1, L_2 son reg $\Rightarrow L_1 L_2$ es reg.)

(Análogo al anterior pero con $r_3 = r_1 r_2$)

Cerrado respecto a la clausura de Keene

(Usando $r_3 = r_1^*$...) (1), (2) y

Cerrado respecto a la complementación:

"Si $L \subseteq \Sigma^*$ es un LR, $\Rightarrow \bar{L} \subseteq \Sigma^*$ es LR tb "

$$\bar{L} = \{ \alpha \in \Sigma^* : \alpha \notin L \}$$

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ tq $\mathcal{L}(M) = L$

Para cada L lng. reg. \exists al menos un AFD M tq $\mathcal{L}(M) = L$

Sea $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$; Veamos que sucede que:

$\forall \alpha \in \Sigma^* ; M \text{ acepta } \alpha \Leftrightarrow M' \text{ no acepta } \alpha$

$M' \text{ acepta } \alpha \Leftrightarrow M \text{ no acepta } \alpha$

$\Rightarrow M'$ define al lenguaje \bar{L} ! ■ Un AFD solo define lenguajes regulares

(1) y (2)
y por def.
de los
expr.
regulares

85
Cerrado respecto a $\boxed{\cap}$

$$L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\overline{L_1} \cup \overline{L_2}} \quad \left. \begin{array}{l} \text{Ley de Morgan} \\ \text{había llegar con} \end{array} \right\} \text{L7}$$

Ahora: ~~manejamos~~

$$\overline{L_1}, \overline{L_2} \text{ es reg} \Rightarrow \overline{L_1} \cup \overline{L_2} \text{ es reg} \Rightarrow \overline{\overline{L_1} \cup \overline{L_2}} \text{ es reg!}$$

↳ L7 uso lo anterior

Algebra de Boole

Los LR son un Algebra de Boole porque son cerrados con respecto a unión, intersección y complemento.

• La \cup y \cap $\boxed{\text{finito}}$ de leng. regulares es un leng. regular

Dem: inducción en la cant. de ~~los~~ participantes a la \cup o \cap .

• **OJO!** La $\boxed{\cup_{\infty}}$ NO es regular siempre. Ej:

$$L_i = \{a^i b^i\}$$

$$\bigcup_{i=1}^{\infty} L_i \text{ NO es regular} = \{a^k b^k : k \in \mathbb{N}\}$$

• Un $\boxed{\text{subconjunto}}$ de un LR, NONEC es LR

PROPIEDADES LRS (2)

- Todo lenguaje finito es regular

Dem: La \cup finita es regular, y puede crear leng. regulares con una sola cadena... \Rightarrow

Problemas decidibles sobre LRS

Pertenencia (Si $\alpha \in L$)

Se construye el AFD asociado a L y se ve si α pertenece.

Finitud:

L es finito, si para su AFD M vale lo siguiente:

" $\forall q \in Q, q$ alcanzable $\Rightarrow q$ puede llegar a un est. final $\Rightarrow q \xrightarrow{*} q$ "

(L es finito si para todo q alcanzable y desde el cual se puede llegar a un estado final, no hay ciclos que empiecen y terminen en q)

 L es infinito si $(M \cap L(M) = L)$

$\exists q \in Q, q$ alcanzable, $q \xrightarrow{*} f \in F \wedge q \xrightarrow{*} q$

 L es infinito si acepta una cadena α tal

$$2n > |\alpha| \geq n \quad ; \quad n = |Q|$$

Verdad:

Se determina el conj. de estados alcanzables A

Si $A \cap F = \emptyset \Rightarrow L$ es vacío

Equivalencia: $(L_1 \equiv L_2)$

Si el lang.

$$(L_1 \cap \bar{L}_2) \cup (L_2 \cap \bar{L}_1) \text{ es } \emptyset$$

$$\Rightarrow L_1 \equiv L_2$$

PUMPING (1)

Defs:

$$\bullet \text{ CI: } (q, \alpha) \in Q \times \Sigma^*$$

$$\bullet \vdash: \vdash: Q \times \Sigma^* \rightarrow Q \times \Sigma^*$$

$$\left[\begin{array}{l} (q, \alpha) \vdash (p, \beta) \\ \text{sii} \\ \alpha = a\beta \quad (a \in \Sigma) \\ \delta(q, a) = p \end{array} \right]$$

$$\bullet \vdash^*: (q, \alpha) \vdash^* (p, \beta) \text{ sii } \alpha = w\beta \wedge \hat{\delta}(q, w) = p$$

Teorema intermedio para el lema de pumping:

$$\text{Si: } (q, \alpha\beta) \vdash^* (q, \beta) \Rightarrow \forall i \geq 0, (q, \alpha^i\beta) \vdash^* (q, \beta)$$

(o sea, si hay un ciclo con origen y fin en 'q', que se recorre consumiendo 'α', se puede recorrer 'i' veces!)

Dem:

Ideas de la demostración:

a) Inducción en i

b) Notar que para $i \geq 1$, $\alpha^i\beta$ puede ser visto como $\alpha \underbrace{\alpha^{i-1}\beta}_{\beta'}$

O sea:

$$\text{Para } i=0, (q, \alpha^0\beta) = (q, \beta) \vdash^* (q, \beta)$$

Sup. válido $\forall i < n$, app $i=n$; $n > 0$

$$(q, \alpha^n\beta) = (q, \alpha \underbrace{\alpha^{n-1}\beta}_{\beta'}) \vdash^* (q, \alpha\beta) \vdash^* (q, \beta) \quad \blacksquare$$

por el
entrecabete
de la hip. del
teorema: "Si $(q, \alpha\beta) \vdash^* (q, \beta) \Rightarrow \dots$ "

PUMPING (2)

LEMA DE PUMPING

Sea L un LR.

\exists una long. mínima de cadenas, " n " tq todas las cadenas mayores a esa longitud pueden ser escritas de la forma

$$z = uvw$$

donde:

- $n \geq |uv|$
- $|v| \geq 1$
- $\forall i \geq 0, uv^i w \in L$

○ sea:

Si L es LR $\Rightarrow \exists n, \forall z, (z \in L \wedge |z| \geq n)$

$$\Rightarrow [z = uvw \wedge n \geq |uv|, |v| \geq 1, \forall i \geq 0, uv^i w \in L]$$

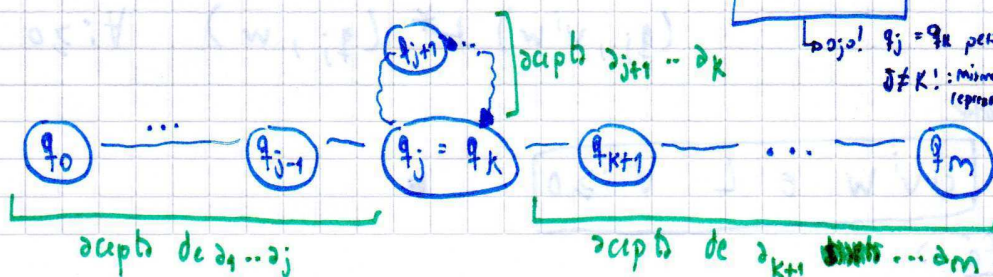
DEM

Tomamos n como la cant. de est. del AFD M tq $L(M) = L$.

Tomamos z , de long. m , tq $m \geq n$ y $z \in L$.

$$z = a_1 \dots a_m$$

Sean q_0, q_1, \dots, q_m la seq. de estados que se recorren para aceptar la cadena z . Como $m \geq n$, $\Rightarrow \exists j, k$ tq $q_j = q_k$ y necesariamente $k \leq n$.



¡¡¡ojo! $q_j = q_k$ pero $j \neq k$! : mismo estado, representado con difs índices

(pensar pq necesariamente es así \Rightarrow)

Particiones z a 3 : $z = uvw$ tq

- $u = \begin{cases} a_1 \dots a_j & \text{si } j > 0 \\ \lambda & \text{si } j = 0 \end{cases}$
- $v = a_{j+1} \dots a_k$
- $w = \begin{cases} a_{k+1} \dots a_m & \text{si } k < m \\ \lambda & \text{si } k = m \end{cases}$

Veremos que:

$$|uv| = k, \text{ y } k \leq n \Rightarrow |uv| \leq n$$

$$|v| \geq 1, \text{ por } \boxed{k > j}$$

Falta probar que $\forall i \geq 0, uv^i w \in L$:

$$\text{Segue } (q_0, uvw) \vdash^* (q_m, \lambda) ; q_m \in F$$

En part.:

$$(q_0, uvw) \vdash^* (q_j, vw) \vdash^* (q_k, w) \vdash^* (q_m, \lambda)$$

Pero $q_j = q_k \Rightarrow$
repr. el mismo estado

$$(q_j, vw) \vdash^* (q_j, w)$$

$$(q_j, v^i w) \vdash^* (q_j, w) \quad \forall i \geq 0$$

\Rightarrow

Teorema

\Rightarrow

$$\boxed{uv^i w \in L \quad \forall i \geq 0}$$

salto de fé \Rightarrow

PUMPING (3)

En realidad este salto de f_i es:

$$\textcircled{1} \quad (q_0, \underbrace{uvw}_{\beta}) \vdash^* (q_j, \underbrace{vw}_{\beta})$$

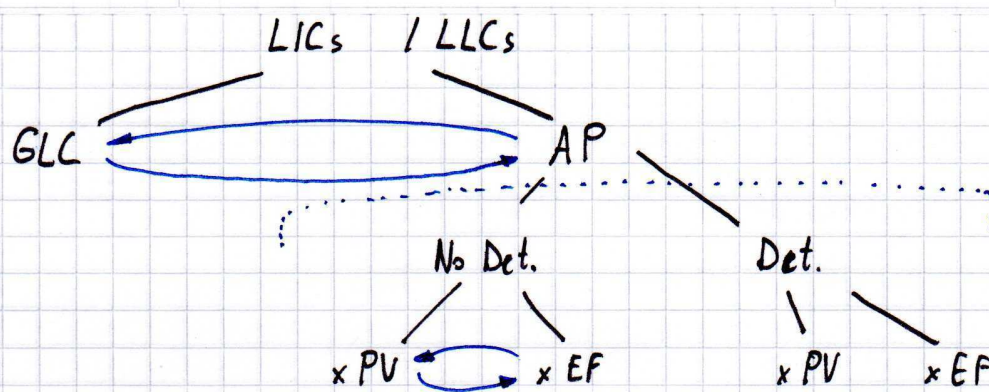
o sea "se consume 'u' entre q_0 y q_j "

$\textcircled{2}$ Luego, tanto si $\beta = vwr$ o $\beta = v^i w$, por el teo anterior,
"se consume $v^i w$, $\forall i \geq 0$
entre q_j y q_k (pq $q_j = q_k$)

y de $\textcircled{1}$ y $\textcircled{2}$ se desprende que $uv^i w \in L$.

=====

Aclaración sobre el salto de f_i



Gramáticas libre de Contexto (glc)

Tiene los prods de la forma:

$$A \rightarrow \alpha$$

donde:

$$A \in V_N$$

$$\alpha \in (V_N \cup V_T)^*$$

Si $\alpha \in (V_N \cup V_T)^+$ (o sea, \exists producción de la forma $A \rightarrow \lambda$)
es una GLC "propia"

Def: Arbol de derivación: (dada una glc G y una cadena α / $\alpha \in L(G)$)

- Cada vértice posee un etiq. a el $\in V_N \cup V_T \cup \{\lambda\}$
- Si un vértice es interior, su etiq. $\in V_N$
- El vértice raíz tiene como etiq. al simb. dist. S ($\in V_N$)
- Si un vértice es hoja, su etiq. $\in V_T \cup \{\lambda\}$
- " " " tiene la etiq. $\lambda \Rightarrow$ es hoja y además, es el único hijo de su padre.
- Si un vértice con etiq. A tiene n hijos $1..n$ tq las etiq. de sus hijos son X_1, \dots, X_n ; $\Rightarrow \exists$ en G una producción de la forma:

$$A \rightarrow X_1 \dots X_n$$

↑
IMPORTANTE,
el AD es por
cadena!!!

Si $G =$

$$\begin{aligned} S &\rightarrow B \\ B &\rightarrow ac \\ B &\rightarrow bd \end{aligned}$$

¿Cómo es el AD?

RTA: EL A.D. es POR CADENA, no
por gramática, entonces hay un AD para
"ac" y otro AD para "bd"

Def. Gramáticas Ambiguas:

Una glc. es ambigua si:

$\exists \alpha, \alpha \in L(G)$ tal que α posee + de un AD asociado.

Def. Leng. ind. de contexto intrínsecamente ambigua:

Si $\forall G: \text{glc}, L = L(G) \Rightarrow G$ es ambigua

(No es posible construir una glc no ambigua para L)

Def. Deriv. + a la izq. (\xrightarrow{L})

Siempre elige el VN más a la izq. a la sustra para el reemplazo:

$\alpha_1 A \alpha_2 \xrightarrow{L} \alpha_1 \beta \alpha_2$ si $\alpha_1 \in V_T^*$ (y $A \rightarrow \beta \in P$)

No hay no terminales a la izq. de A

Def. Deriv. + a la derecha (\xrightarrow{R})

Análogo...

$\alpha_1 A \alpha_2 \xrightarrow{R} \alpha_1 \beta \alpha_2$ si $\alpha_2 \in V_T^*$ ($A \rightarrow \beta \in P$)

En un árbol ~~completo~~ completo y balanceado donde todos los vertices no-hoja tienen el mismo degree:

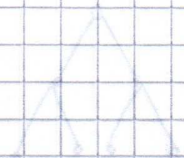
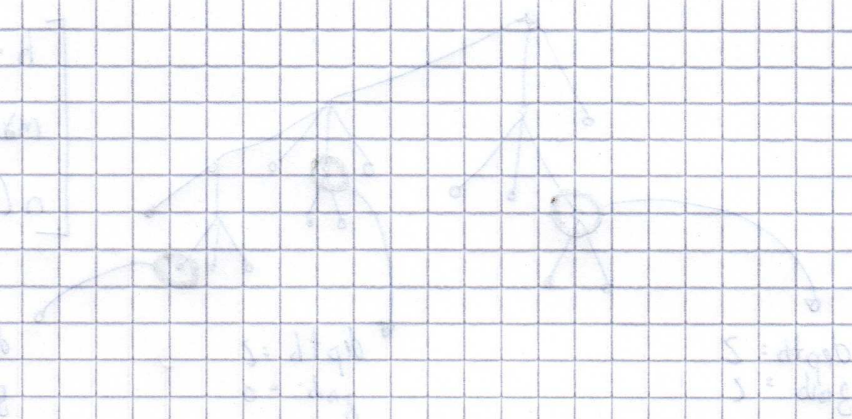
$$\text{cant. hojas} = b^h$$

$h = \text{height}$

$b = \text{degree del árbol.}$

Si no se cumple que sea completo, balanceado y con igual degree, entonces

$$\text{cant. hojas} \leq (\text{max. degree})^h$$



$$1 - \frac{1}{b} < \frac{1}{b} < 1 - \frac{1}{b}$$

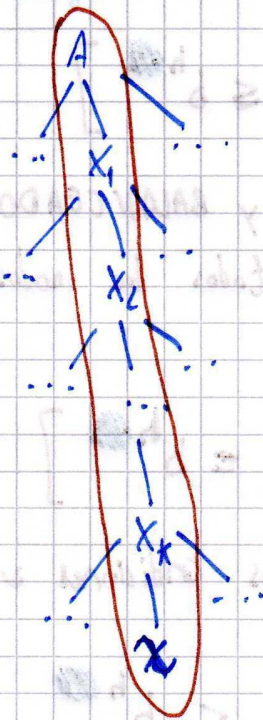
Def. sobre árboles: (sin ocuparnos de la glc asociada a la hora de estas defs...)

- Camino: Camino de X en un árbol $T(A)$ - donde X es un nodo a la secuencia A, X_1, \dots, X_k, X tq:

$$A \rightarrow \dots X_1 \dots \rightarrow \dots X_2 \dots \rightarrow \dots \rightarrow \dots X_k \dots \rightarrow \dots X \dots$$

donde:

$$A, X_1, \dots, X_k \in T(A)$$



donde:

$$A \rightarrow \dots X_1 \dots$$

$$X_1 \rightarrow \dots X_2 \dots$$

$$X_2 \rightarrow \dots X_3 \dots$$

\vdots

$$X_k \rightarrow \dots X \dots$$

Todas estas
producciones
pertenecen a P
de en conjunto

- Altura de un árbol $T(A)$

$\max\{\{ | \alpha X | \} ; \text{ donde } X \text{ es la etiq. de una hoja y } \alpha X \text{ es camino de } T(A) \}$
 No cuenta la # de caracteres, sino los arcos

- Longitud de un camino de X : la cantidad de ARCOS del camino que une A con X en $T(A)$

Se lee así:
 "La long. de la cadena α derivada del AD $T(S)$ asociado a G , es menor o igual a la máx. long. de la parte derecha de cada producción en P , elevada a la altura del árbol"

Lema:

Sea G una glc, $\langle V_N, V_T, P, S \rangle$, $P \neq \emptyset$.

Sea $T(S)$ un AD en G para α , de altura ' h '

Si $a = \max \{ |\beta| : A \rightarrow \beta \in P \}$ entonces $|\alpha| \leq a^h$

DEM

en carpeta de SOLE

A ver:

(Dem. a lo carpeta de SOLE, pero > es una alternativa).

Para cualquier árbol tal que:

- altura = h
 - cont. de ramas por nodo no-hoja = b (degree)
- vale que: $\left[\text{cont. de hojas máximas} \leq b^h \right]$

Si: el árbol es COMPLETO y BALANCEADO (todos los caminos raíz-hoja tienen igual longitud y todos los nodos internos tienen b ramificaciones) vale que:

$$\left[\text{cont. de hojas} = b^h \right]$$

Para si alguna de las dos condiciones anteriores no se cumplen, entonces:

$$\text{cont. de hojas} < b^h$$

En nuestro lema:

$$|\alpha| \geq \max \{ b \}$$

(depende de si en la deriv. de α en ese AD se usa o no la producción $A \rightarrow \beta$ donde β es de long. máxima...)

$$|\alpha| = \text{cont. de hojas}$$

$$\Rightarrow |\alpha| \leq a^{h-1} \leq a^h \dots \text{¿Está bien?}$$

¡¡PARECE MUY SIMPLE!! ~> ¿está bien?

PUMPING (GRAMÁTICAS LIBRES DE CONTEXTO)

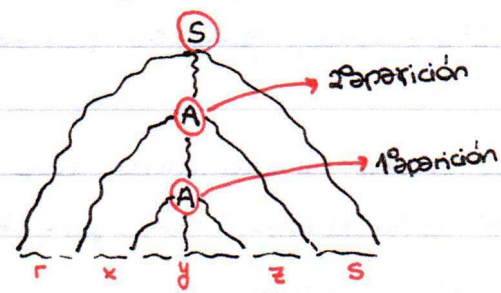
$\cdot |xyz| \leq n$
 $\cdot |x| \geq 1$
 $\cdot x^i y z^i \in L$

Lema: $\forall L$ independiente de contexto, $\exists m > 0$:

$$\forall \alpha \in L, (|\alpha| \geq m \Rightarrow \exists r, x, y, z, s : \alpha = rxyzs \wedge |xyz| \leq m \wedge |x| \geq 1 \wedge \forall i \geq 0, (rxy^iz^is \in L)).$$

Dem: Sea G una g.c. tal que $L = L(G)$ y sea $a = \max \{ |p| : a \rightarrow p \in P \}$. Tomamos $m = a^{|V_N|+1}$ consideremos la cadena $\alpha \in L$ tal que $|\alpha| \geq m$. Sea $T(s)$ un árbol mínimo (de altura mínima) de derivación de α . Por el lema previo, resulta que $a^h \geq |\alpha|$. Con lo cual, $a^h \geq |\alpha| \geq m = a^{|V_N|+1}$.

De allí, se desprende que $h \geq |V_N| + 1$. Entonces, existirá algún símbolo en α tal que su camino será de longitud mayor o igual a $|V_N| + 1$. Como la cantidad de símbolos no terminales es $|V_N|$, entonces en ese camino seguramente existe un no terminal repetido, llamado A :



La segunda aparición de A da lugar a la cadena xyz . Como tenemos la garantía de que podemos encontrar un A tal que esa segunda aparición esté a una distancia no mayor de $|V_N| + 1$ respecto de la base, entonces es cierto que:

$$m = a^{|V_N|+1} \geq |xyz|.$$

Por otro lado, si $|xz|$ fuera 0, el árbol podría recortarse, pues al ocurrir $A \rightarrow A$, se reduciría en al menos 1 unidad la longitud del camino.

Si este camino fuera el de longitud mínima, se reduce la altura del árbol, el cual supusimos que era mínimo de altura mínima.

Si no, se efectúa el mismo razonamiento con el árbol que resulta de recortar ese camino máximo. Este árbol también derivaría α , con lo cual se cumple: $|xz| > 0$

Por último, ya por inducción, falta ver que $\forall i \geq 0, rxy^iz^is \in L$.

Caso Base: es trivial que $S \xRightarrow{*} rAS \xRightarrow{*} ryS \Rightarrow rxy^0z^0s = ryS \in L$

P.I.: $S \xRightarrow{*} r \underbrace{x^{i-1}}_{\text{por H.I.}} \underbrace{xyz}_{\text{trivial}} z^{i-1} s \Rightarrow rxy^iz^is \in L$

NOTA

Por ejemplo, probaremos que $L = \{0^m 1^m 2^m \mid m \geq 1\}$ no es independiente del contexto:

Supongamos que L fuera independiente de contexto, entonces debería existir un m entero dado. elegimos $z = 0^m 1^m 2^m$.

Supongamos partir z en $uvwx$, en donde $|vwx| \leq m$ y v y x no son ambos ϵ . Entonces sabemos que vwx no puede tener 0's y 2's ya que el último 0 y el primer 2 están a $m+1$ posiciones de distancia.

Intentaremos ver que L contiene un string que se sabe que no pertenece a L , contradiciendo la hipótesis de que L es independiente de contexto. Se pueden dar los casos:

- vwx no tiene 2's \Rightarrow entonces, vx solo tiene 0's y 1's, y tiene por lo menos uno de estos símbolos. luego, uv^2wx , quien debería pertenecer a L por Pumping, tendría m 2's pero menos de m 0's o 1's, o menos de ambos. luego, no pertenece a L , y en este caso L no es independiente de contexto
- vwx no tiene 0's: de manera similar, uv^2wx tiene m 0's (por u) y menos cantidad de 1's o 2's. luego, tampoco pertenece a L .

En ambos casos, concluimos en que L contiene una cadena válida que no pertenece a L . Esta contradicción nos permite concluir que L no es un lenguaje libre de contexto.

El juego:

- El adversario elige m . No lo conocemos \Rightarrow planeamos por todo posible m
- elegimos $|cadena| \geq m$, podemos usar m como parámetro
- El adversario divide cadena en $uvwx$, en la cual $|vwx| \leq m$ y $vx \neq \epsilon$
- Ganamos si podemos, eligiendo algún i y mostrando que uv^iwx^iy no \in el lenguaje!

PROPIEDADES DE LENGUAJES LIBRES DE CONTEXTO

Teorema: Si L_1 y L_2 son dos LIC, entonces $L_1 \cup L_2$ lo es también.

Dem: Como L_1 y L_2 son LIC, entonces existen $G_1 = \langle V_{N1}, V_{T1}, A_1, S_1 \rangle$ y $G_2 = \langle V_{N2}, V_{T2}, A_2, S_2 \rangle$ gramáticas independientes de contexto tales que $L_1 = L(G_1)$ y $L_2 = L(G_2)$.

Supongamos, sin perder la generalidad, que $V_{N1} \cap V_{N2} = \emptyset$. Definimos:

$$G = \langle V_{N1} \cup V_{N2} \cup \{S\}, \Sigma, A \cup B \cup \{S \rightarrow S_1 \mid S_2\}, S \rangle$$

y puede demostrarse fácilmente que $\forall d \in \Sigma^*, d \in L(G) \Leftrightarrow d \in L(G_1) \cup L(G_2)$.

Teorema: Si L_1 y L_2 son dos LIC, entonces $L_1 L_2$ lo es también. ~~Debe demostrarse si~~

Dem: Procediendo de manera similar a lo anterior, definimos:

$$G = \langle V_{N1} \cup V_{N2} \cup \{S\}, \Sigma, A \cup B \cup \{S \rightarrow S_1 S_2\}, S \rangle.$$

Teorema: Si L es LIC, entonces L^+ y L^* , también.

Dem: Definimos respectivamente,

$$G = \langle V_{N1} \cup \{S\}, \Sigma, A \cup \{S \rightarrow S_1 S, S \rightarrow S_1\}, S \rangle$$

$$G = \langle V_{N1} \cup \{S\}, \Sigma, A \cup \{S \rightarrow S_1 S, S \rightarrow \lambda\}, S \rangle.$$

Teorema: Si L_1 y L_2 son dos LIC, $L_1 \cap L_2$ no es necesariamente LIC.

Dem: Sean $L_1 = \{a^m b^m c^m : m, m \geq 0\}$ y $L_2 = \{a^m b^m c^m : m, m \geq 0\}$, la intersección $L_1 \cap L_2 = \{a^m b^m c^m : m \geq 0\}$ el cual probamos que no es LIC.

Puede verse fácilmente que L_1 y L_2 son LIC, además. por ejemplo, para L_1 :

$$G_1 = \langle \{S, A, C\}, \{a, b, c\}, \{S \rightarrow AC \mid C, A \rightarrow aAb \mid ab, C \rightarrow cC \mid \lambda\}, S \rangle$$

Teorema: $L = \{ww : w \in \Sigma^*\}$ no es LIC

Dem: Se deriva del Teorema que veremos más adelante cuando veamos AD's, que dice que la intersección entre un lenguaje regular y uno libre de contexto da como resultado uno libre de contexto. ~~Veremos~~ ^{Se} $L_2 = \{a^m b^m a^m b^m : m, m \geq 0\}$

~~Veremos~~ $L_1 = L \cap a^* b^* a^* b^*$. L_1 no es LIC, por ende L tampoco lo es.

La demostración que complementa a esto, mencionada, se encuentra en pág. 48.

esto es $\{ww\}$

esto es regular

esto es $\{a^n b^n a^n b^n\}$; y no es LIC (se demuestra por pumping)

es por lo tanto como L_1 no es LIC y L_2 es LIC

Intersección entre LR y LIC

Teorema: no es cierto que si L es IC entonces \bar{L} es IC.

Dem: Supongamos

\bar{L}_1 y \bar{L}_2 son IC.

Como la unión es IC, entonces $\bar{L}_1 \cup \bar{L}_2$ es IC. Suponiendo que \bar{L} es IC, $\overline{L_1 \cup L_2}$ debería serlo, y $\overline{L_1 \cup L_2} = \bar{L}_1 \cap \bar{L}_2$ el cual no es necesariamente IC, ~~pero~~ pues ya lo demostramos.

→ Si!

• $L - R$? = $L \cap \bar{R} \Rightarrow L - R$ es un LIC.

Esto es regular.

→ no!

• $L_1 - L_2$? = $L_1 \cap \bar{L}_2$

Σ^* es IC para todo alfabeto Σ . Luego, si $L_1 - L_2$ fuera IC siempre cuando L_1 y L_2 lo son, ocurriría que $\Sigma^* - L$ sería IC siempre que L lo fuera. pero estamos hablando de \bar{L} en ese caso! ($\bar{L} = \Sigma^* - L$) Esto contradice a la prop de \bar{L} .

→ Si!

• L^R ? Sea $L = L(G)$ para alguna CFL $(G) = \langle V, T, P, S \rangle$.

Construimos $G^R = \langle V, T, P^R, S \rangle$ en donde P^R es el "reverso" de cada producción en P . es decir, $A \rightarrow \alpha \in P \Rightarrow A \rightarrow \alpha^r \in P^R$.

Se ve fácil x inducción en la ley de las derivaciones en G y G^R de que $L(G^R) = L^R$.

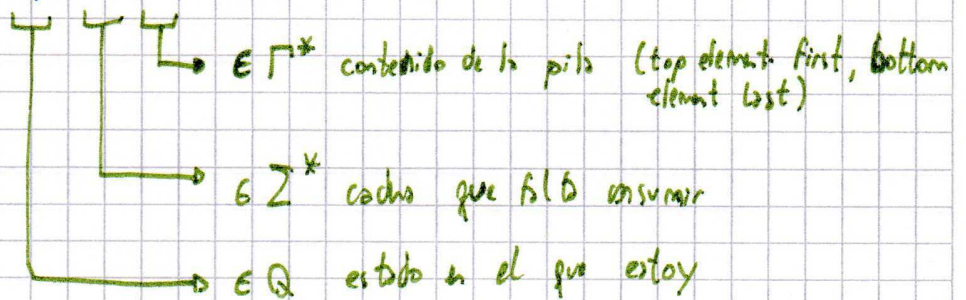
APND

M es un APND:

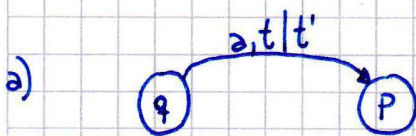
$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$$

- Q = Conj. de estados
- Σ = Alfabeto de entr. finito
- Γ = " de pila, finito
- $\delta: Q \times (\Sigma \cup \lambda) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$
- q_0 : estado inicial
- $Z_0 \in \Gamma$: Conf. inicial de pila.
- $F \subseteq Q$

C.I. de un APND: $\langle q, \alpha, \pi \rangle$



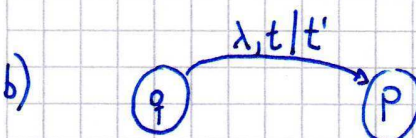
Relación entre una CI y la función δ



$$\langle q, \alpha, t, \pi \rangle \vdash \langle p, \alpha, t', \pi \rangle$$

sii

$$(p, t') \in \delta(q, a, t)$$



$$\langle q, \alpha, t, \pi \rangle \vdash \langle p, \alpha, t', \pi \rangle$$

sii

$$(p, t') \in \delta(q, \lambda, t)$$

Cadenas aceptadas:

$$L \underset{x \in PV}{\in} M \quad \text{si} \quad \langle q_0, \alpha, z_0 \rangle \vdash^* \langle p, \lambda, \lambda \rangle$$

$$L \underset{x \in F}{\in} M \quad \text{si} \quad \langle q_0, \alpha, z_0 \rangle \vdash^* \langle p, \lambda, \pi \rangle \quad \text{y} \quad p \in F$$

Un APND acepta 2 lenguajes ~~definidos~~:

$$\begin{cases} L(M) &= \{ \alpha_j \mid x \in F \\ L_\lambda(M) &= \quad \quad \quad x \in PV \end{cases}$$

$$\mathcal{L}(M) \Rightarrow \mathcal{L}_\lambda(M)$$

Teorema: Si $L = \mathcal{L}(M)$ donde $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ entonces existe M' tal que

$$\mathcal{L}_\lambda(M') = \mathcal{L}(M) = L. \text{ Definamos } M':$$

Def: $M' = \langle Q \cup \{q_\lambda, q'_\lambda\}, \Sigma, \Gamma \cup \{\lambda\}, \delta', q'_\lambda, X_0, \emptyset \rangle$

En donde:

$$1. \delta'(q'_\lambda, \lambda, X_0) = \{(q_0, Z_0 X_0)\}$$

$$2. \forall q \in Q, \forall a \in \Sigma \cup \{\lambda\}, \delta'(q, a, Z) = \{(r, \gamma) : (r, \gamma) \in \delta(q, a, Z) \text{ con } r \in Q \wedge \gamma \in \Gamma^*\}$$

$$3. \forall q \in F, \forall Z \in \Gamma \cup \{\lambda\}, (q_\lambda, \lambda) \in \delta'(q, \lambda, Z)$$

$$4. \forall Z \in \Gamma \cup \{\lambda\}, (q_\lambda, \lambda) \in \delta'(q_\lambda, \lambda, Z)$$

$\{q_\lambda\}$ parece ser un estado especial que VACÍA la pila.
 $\{q'_\lambda\}$ parece ser un nuevo est. inicial después ANTES de qo... ver regla 1

des de cualquier est. final de M , si la columna se puede ir a " q_λ "; (3)
 y " q_λ " vacío la pila. (4)

- Con la regla 1 tenemos que M' entra al estado inicial del autómata M : q_0 , con $Z_0 X_0$ en la pila. De manera que si por una cadena cualquiera, M intentara vaciar la pila desapilando Z_0 , sin existiría X_0 , provocando que la pila no se vacíe.

- La regla 2 implica que M' simula a M .

- De las reglas 3 y 4, si M entra en un estado final, siempre puede elegir entre ir al estado q_λ y vaciar la pila o bien continuar simulando M .

es una opción, no una obligación

Veamos que $\mathcal{L}(M) = \mathcal{L}(M')$.

$$\Leftarrow: x \in \mathcal{L}(M) \Rightarrow (q_0, x, Z_0) \xrightarrow{*}_M (q, \lambda, \gamma) \wedge q \in F \text{ por def. de } \mathcal{L}(M).$$

• por regla 1, $(q'_\lambda, \lambda, X_0) \xrightarrow{*}_{M'} (q_0, x, Z_0 X_0)$

• por regla 2, $(q_0, x, Z_0) \xrightarrow{*}_{M'} (q, \lambda, \gamma)$

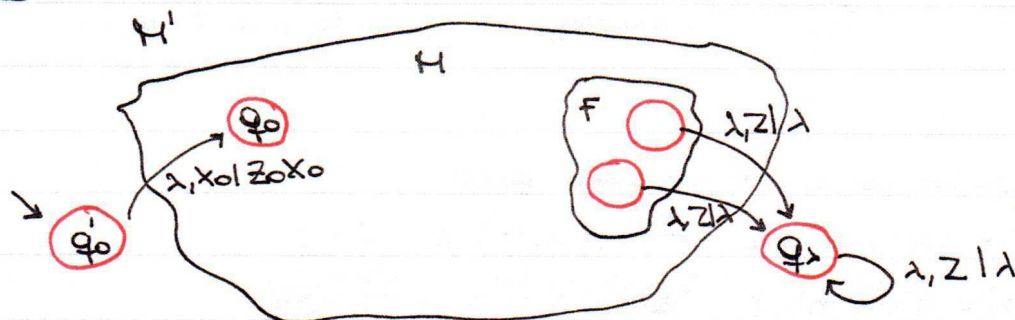
reglas 3 y 4

Entonces, $(q'_\lambda, \lambda, X_0) \xrightarrow{*}_{M'} (q_0, x, Z_0 X_0) \xrightarrow{*}_{M'} (q, \lambda, \gamma X_0) \xrightarrow{*}_{M'} (q_\lambda, \lambda, \lambda) \Rightarrow x \in \mathcal{L}_\lambda(M')$.

$$\Rightarrow: x \in \mathcal{L}_\lambda(M'), \Rightarrow (q'_\lambda, \lambda, X_0) \xrightarrow{*}_{M'} (q_0, x, Z_0 X_0) \xrightarrow{*}_{M'} (q, \lambda, \gamma X_0) \xrightarrow{*}_{M'} (q_\lambda, \lambda, \lambda)$$

$$(q_0, x, Z_0) \xrightarrow{*}_M (q, \lambda, \gamma) \Rightarrow x \in \mathcal{L}(M)$$

Gráfico:



$$\boxed{\mathcal{L}_\lambda \Rightarrow \mathcal{L} \text{ (AP)}}$$

Teorema: Si $L = \mathcal{L}_\lambda(M')$ en donde $M' = \langle Q', \Sigma, \Gamma', \delta', q'_0, \overline{x_0}, F' \rangle$ entonces existe M tal que $\mathcal{L}(M) = L$, definido como $M = \langle Q \cup \{q_0, q_f\}, \Sigma, \Gamma' \cup \{Z_0\}, \delta, q_0, \overline{Z_0}, \{q_f\} \rangle$

En donde :

$$\textcircled{1} \delta(q_0, \lambda, Z_0) = \{ (q'_0, x_0 Z_0) \}$$

$$\textcircled{2} \forall q \in Q, \forall a \in \Sigma \cup \{ \lambda \}, \delta(q, a, Z) = \delta'(q, a, Z)$$

$$\textcircled{3} \forall q \in Q, (q_f, \lambda) \in \delta(q, \lambda, Z_0)$$

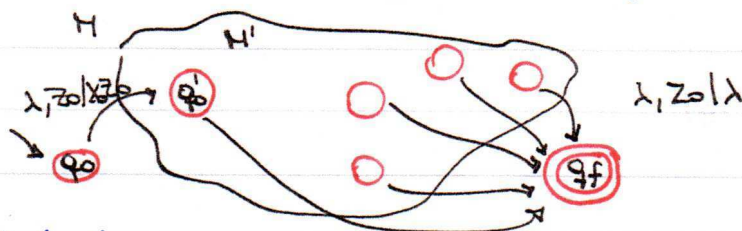
o sea si llego al punto tal que lo único que queda en la pila es lo mismo que tenía cuando entré (Z_0); entonces puede salir a q_f sin importar en qué estado esté.

Dem:

Por regla $\textcircled{1}$, entramos en M' desde el principio, con $x_0 Z_0$ en la pila.

Por regla $\textcircled{2}$, recorremos a M' .

Por regla $\textcircled{3}$ le permite, en cualquier momento que vacíe la pila, salir al estado q_f .



El agregado de Z_0 hace que donde antes la pila se vaciaba, ahora va a que der los Z_0 .

$$(q_0, x, Z_0) \xrightarrow{M} (q'_0, x, x_0 Z_0) \xrightarrow{*}_{M'/M} (q, \lambda, Z_0) \xrightarrow{M} (q_f, \lambda, \lambda) \wedge q_f \in F \text{ por def.}$$

$$\bullet \text{ Si } x \in \mathcal{L}(M') \Rightarrow (q'_0, x, Z_0) \xrightarrow{*}_{M'} (q, \lambda, \lambda).$$

$$\text{Por regla 1), } (q_0, x, Z_0) \xrightarrow{M} (q'_0, x, x_0 Z_0)$$

$$\text{Por regla 2), } (q'_0, x, x_0 Z_0) \xrightarrow{*}_{M'/M} (q, \lambda, Z_0)$$

$$\text{Por regla 3) } (q, \lambda, Z_0) \xrightarrow{M} (q_f, \lambda, \lambda)$$

con lo cual, ~~se concluye~~

$$(q_0, x, Z_0) \xrightarrow{M} (q'_0, x, x_0 Z_0) \xrightarrow{*}_{M'} (q, \lambda, Z_0) \xrightarrow{M} (q_f, \lambda, \lambda) \Rightarrow x \in \mathcal{L}(M).$$

Para ver que $x \in \mathcal{L}(M) \Rightarrow x \in \mathcal{L}'(M')$, debemos demostrar ~~los pasos~~ los pasos:

$$(q_0, x, Z_0) \xrightarrow{M} (q'_0, x, x_0 Z_0) \xrightarrow{*}_{M'} (q, \lambda, Z_0) \xrightarrow{M} (q_f, \lambda, \lambda)$$

Es lo que hace computar M' , a diferencia que Z_0 está en la pila. y M' además no puede más que usar esas transiciones. Es como que los movimientos de M' están incluidos en los de M . $\Rightarrow (q'_0, x, x_0) \xrightarrow{M'} (q, \lambda, Z_0) \Rightarrow x \in \mathcal{L}'(M')$.

AUTOMATA DE PILA DETERMINISTICO

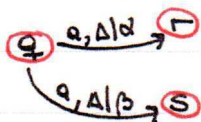
Es un automata de pila $M = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ para el que se cumple que:

~~Es un automata de pila deterministico~~

~~Es un automata de pila deterministico~~

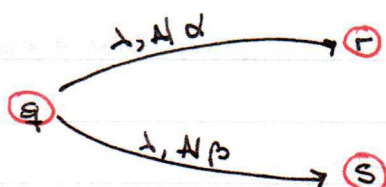
$$① \quad |\delta(q, a, A)| \leq 1$$

o sea, no puede ocurrir:



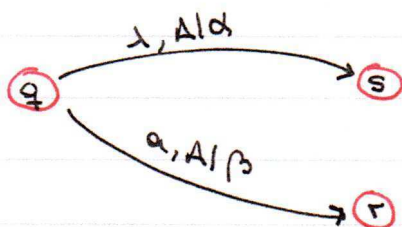
$$② \quad |\delta(q, \lambda, A)| \leq 1$$

o sea, no puede ocurrir:



$$③ \quad \text{si } |\delta(q, \lambda, A)| = 1 \text{ entonces } |\delta(q, a, A)| = 0$$

o sea, no puede ocurrir:



Observaciones: r y s no necesariamente son distintos

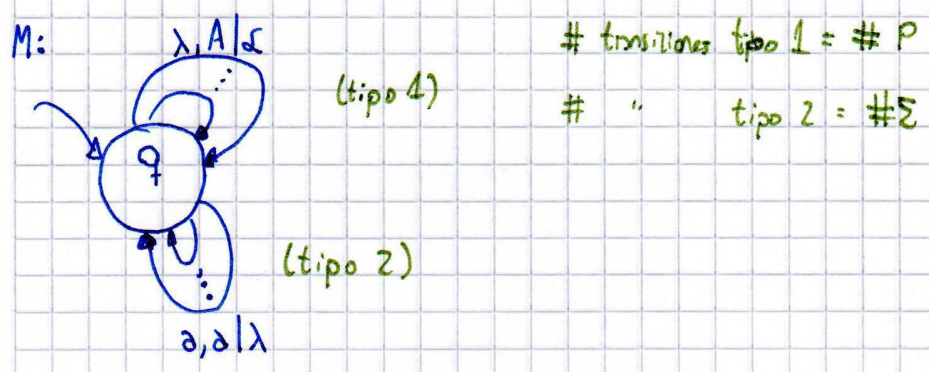
a y b no necesariamente son distintos

GLC \rightarrow APND (1)

Dada una GLC $G = \langle V_N, V_T, P, S \rangle$, construyo M tal q:

$$M = \langle \{q\}, V_T, V_N, \delta, q, S, \emptyset \rangle ; \quad \delta : \begin{cases} \text{Si } A \rightarrow \alpha \in P; (q, \alpha) \in \delta(q, \lambda, A) \\ \forall a \in V_T, \delta(q, a, a) = \{(q, \lambda)\} \end{cases}$$

$t_q \quad \mathcal{L}_\lambda(M) = \mathcal{L}(G)$



Algoritmo:

- 1 WHILE pila no vacía y cadena no vacía:
 - 2 a = caracter a consumir en cadena de entrada
 - 3 t = tope de pila
 - 4 if $t \in (V_T \cup \{\lambda\})$
 - 5 if $t = a$
 - 6 desapila a de la pila y consume a
 - 7 else
 - 8 ERROR, cadena no aceptada
 - 9 else // $t \in V_N$; o sea $t = A$ para algún A
 - 10 desapila A
 - 11 Apilo α ; // $A \rightarrow \alpha \in P$; para alguna producción de esa forma
 - 12 Fi
 - 13 LOOP
- Como M es no-det; dada una cadena, y un estado de consumo, no genera un único estado siguiente, sino potencialmente muchos más!!!

INTERSECCIÓN ENTRE LR y LIC.

Teorema: Si L es un LIC y R es un LR entonces $L \cap R$ es un LIC.

Dem: Sea $P = \langle Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P \rangle$ un AP que acepta L por estado final, y sea $A = \langle Q_A, \Sigma, \delta_A, q_A, F_A \rangle$ un AFD de R .

La idea es correr el AFD en paralelo con un AP, dando como resultado otro AP.

Sea $P' = \langle Q_P \times Q_A, \Sigma, \Gamma, \delta, \{q_P, q_A\}, Z_0, F_P \times F_A \rangle$ en donde:

$\delta((q, p), a, X)$ son todos los pares $((r, s), \gamma)$ tales que

- ①. $s = \delta_A(p, a)$

- ②. $(r, \gamma) \in \delta_P(q, a, X)$.

$$\left. \begin{array}{l} \text{①} \\ \text{②} \end{array} \right\} (a \in \Sigma \cup \{\lambda\})$$

Esto es, por cada movimiento del AP P , podemos hacer el mismo en P' , y

además, se ~~avanza~~ ^{avanza} llevando el estado del ~~AP~~ AFD en la segunda componente de P' .

- cuando $a \in \Sigma$ y muy más cómodo con a , me muevo en ambos ($\delta(p, a) = \delta_A(p, a)$)
- cuando a es λ , me muevo solo en P . ($\delta(p, a) = p$)

Probaremos con una inducción en la cantidad de movimientos del AP P' ,

que $(q_P, w, Z_0) \xrightarrow{*}_P (q, \gamma, \gamma) \iff ((q_P, q_A), w, Z_0) \xrightarrow{*}_{P'} ((q, p), \lambda, \gamma)$, siendo $\gamma = \delta_A(q_A, w)$.

Caso Base: # pasos = 0, $w = \lambda$

Entonces $((q_P, q_A), \lambda, Z_0) \xrightarrow{0}_{P'} ((q_P, q_A), \lambda, Z_0) \iff (q_P, \lambda, Z_0) \xrightarrow{0}_P (q_P, \lambda, Z_0)$, y $q_A = \delta(q_A, \lambda)$ por def. AFD.

Caso Inductivo: # pasos: $i+1$, $w = x a$, con $a \in \Sigma \cup \{\lambda\}$.

$$((q_P, q_A), w, Z_0) \xrightarrow{i+1}_{P'} ((q, p), \lambda, \gamma) \iff ((q_P, q_A), x a, Z_0) \xrightarrow{i}_{P'} ((q', p'), a, \beta) \xrightarrow{1}_{P'} ((q, p), \lambda, \gamma)$$

• Por hipótesis inductiva, sabemos que $\delta_A(q_A, x) = p'$ y $(q_P, x, Z_0) \xrightarrow{i}_P (q', \lambda, \beta)$

• Por def. de δ , $((q', p'), a, \beta) \xrightarrow{1}_{P'} ((q, p), \lambda, \gamma)$

$$\text{Luego, } ① \iff ② \wedge ③ \iff \delta_A(q_A, x a) = p \wedge (q_P, x a, Z_0) \xrightarrow{i+1}_P (q, \lambda, \gamma).$$

y como (q, p) es un estado de aceptación en P' o q es un estado de aceptación de P y p es un estado de aceptación de A , concluimos que P' acepta w o P y A ambos lo hacen, o sea, que $w \in L \cap R$.

PARSERS

Algunas gramáticas 'clave'

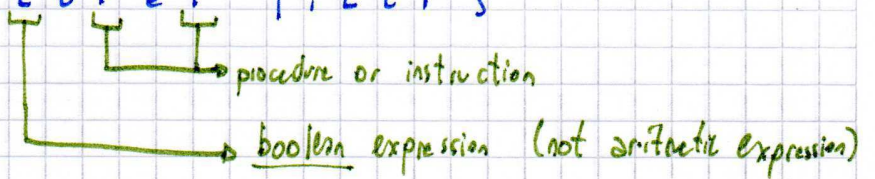
a) Operaciones aritméticas

$$G \text{ con } P = \{ E \rightarrow E + T \mid T, \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid id \mid c \}$$

b) IF-THEN-ELSE ('i', 't', 'e')

~~Gramática para IF-THEN-ELSE~~

$$G \text{ con } P: \{ S \rightarrow i E t P e P \mid i E t P \}$$



Vemos:

- No es RI

- Pero tampoco es LL(1). Es ambiguo:

$i E t (i E t P e P)$

$i E t (i E t P) e P$

Por eso los bloques if-then-else usan delimitadores de bloques ("{" , "}") o reglas de precedencia para evitar esta situación.

Parsers (Descendentes)Primeros

$$\text{Primeros} : (V_N \cup V_T)^* \rightarrow \mathcal{P}(V_T)$$

$$\text{Primeros}(\alpha) = \{ t \in V_T : \alpha \xRightarrow{*} \beta \wedge \beta = t\beta' \}$$

(Primeros aplica sobre parámetros de estado que se considera que son FS.)

Siguientes

$$\text{Siguientes} : V_N \rightarrow \mathcal{P}(V_T)$$

$$\text{Siguientes}(A) = \{ t \in V_T : S \xRightarrow{*} \dots At \dots \}$$

Simbolos Directrices (SD)

$$\text{Simbolos Directrices} : \text{Producción} \rightarrow \mathcal{P}(V_T)$$

$$\text{Simbolos Directrices}(N \rightarrow \beta) = \begin{cases} \text{Primeros}(\beta) & // \text{ si } \beta \text{ no es nullable} \\ \text{Primeros}(\beta) \cup \text{Siguientes}(N) & // \text{ si } \beta \text{ es nullable.} \end{cases}$$

Sobre los parsers descendentes:

→ tienen 4 operaciones expandir
matchear
aceptar
rechazar

- Buscan la derivación + a la izq.
- Van construyendo el árbol de derivación de la raíz a las hojas
- Construyen el árbol en "preorder"
- Con retroceso (No Det.); ^{deriva en V_N} van "probando" a ver que producción pueden usar para expandir
- Sin retroceso (Det.) - Usan los SD para definir que producción usar (útil en V_N). Hay 2 formas
 $\left. \begin{array}{l} \text{pueden} \\ \text{controlar} \end{array} \right\} \text{ambos usan los SD, ambos requieren gramáticas LL(1)}$

Gramáticas extendidas: (o ampliables) - (No conf. con las gramáticas extendidas ELL)

- Se agrega "S" a V_N y ~~se agrega~~ y " $S' \rightarrow S\#$ " a P ($\# \notin V_T$)

A.S. Desc. PredictivoAlgoritmo de A.S. descendente.

Sea $G = \langle V_N, V_T, P, N_i \rangle$ con $V_N = \{N_1, \dots, N_k\}$, considerando $N_i \in V_N$ con las producciones asociadas $N_i \rightarrow d_1 | d_2 | \dots | d_k$. Sea t el símbolo apuntado en w , la cadena de entrada.

\exists un proc. de tipo 'P' como el de abajo, para cada no-terminal en G . (" P_{N_i} ")

$P: \text{cadena} \times V_N \rightarrow \text{bool}$

(\emptyset sea, si hay $\# V_N$ no-terminales, se puede ver como que existen $\#$ procs. 'P'; o también como que existe un proc. P que toma como Z : parámetro a un no terminal.)

Procedimiento P (con backtracking)

Para $l \leftarrow 1 \dots K$

$w' \leftarrow w$

$\text{error} \leftarrow \text{procesar}(w', N_i \rightarrow d_l)$

Si $\neg \text{error}$

devuelve $\neg \text{error}$

Fin Si

Fin Para

devuelve error

$P: \frac{w}{\text{cadena}} \times \frac{N_i}{V_N} \rightarrow \text{bool}$

// Hay K prods de la forma $N_i \rightarrow d_l$

// w y w' son cadenas

// N_i es el no terminal

\vdots K todos los caminos posibles \vdots

// Este procedimiento sencillamente, cubre que

// hay error, prueba con la prod. ($N_i \rightarrow d_l$)

// siguiente, hasta que se le acaban. Y ahí

// si devuelve error

Procedimiento P (según A.S. descendente predictivo) $P: \text{cadena} \times V_N \rightarrow \text{bool}$

Según t

$t \in \text{SD}(N_i \rightarrow d_1) : \text{procesar}(w, N_i \rightarrow d_1)$

:

$t \in \text{SD}(N_i \rightarrow d_k) \text{ procesar}(w, N_i \rightarrow d_k)$

fin según

Procedimiento Procesador ($w', N \rightarrow d$) (con t' apuntando al 1º símbolo de w' , $d = x_1 \dots x_n$)

```

1  Si  $d = \lambda$  {
2      devolver  $\text{error} \leftarrow t' \neq \#$ 
3  } Si no {
4      Para  $j = 1 \dots m$  {
5          Si  $x_j \in V_T$  {
6              Si  $t' = x_j$  {
7                  avanzar puntero en  $w'$ 
8              }
9          } Si no {
10             devolver  $\text{error} \leftarrow \text{true}$ 
11         }
12     } Si  $x_j \in V_N$  {
13         error  $\leftarrow P(w', x_j)$ 
14     }
15 }
```

Operación "matchs (x_j)":
matchs el V_T x_j a la entrada
apuntada por t' , y avanza t

Operación "expandir": encontrar
un V_N y va a seleccionar una
producción de la forma $x_j \rightarrow \dots$
que sea adecuada, llamando al proc. P
asociado a x_j .

→ idea: que w' y d hagan matchs. en caso
de que x_j sea un no terminal, se llama a P .

Programa principal (main)

```

{
     $P(w', N_1)$  //  $N_1$  es 'S' el símbolo distinguido
    Si  $\text{error} \rightarrow$  la cadena no pertenece al lenguaje
}
```


Anál. Sint. Desc. Predict. Sin Tabla (Ejemplo)

Vamos un ejemplo:

$S \rightarrow \text{Tipo } \$$
 $\text{Tipo} \rightarrow \text{Simple} \mid \uparrow \text{Simple} \mid \text{array} [\text{Simple}] \text{ of Tipo}$
 $\text{Simple} \rightarrow \text{int} \mid \text{char} \mid \text{num}.. \text{num}$

→ Uso las gramáticas ampliado :)
 // se puede usar \$ o #

6

$V_T = \{ \text{int, char, num, .., [,], array, of, } \uparrow \}$

$V_N = \{ S, \text{Tipo, Simple} \}$

Primeros:

| | |
|--------------------------|----------------|
| Simple | int, char, num |
| $\uparrow \text{Simple}$ | \uparrow |
| array [Simple] of Tipo | array |
| int | int |
| char | char |
| num..num | num |

SD (Para cada producción)

| Producción | variable? | SDs |
|--|-----------|--|
| $S \rightarrow \text{Tipo } \$$ | 1 | $\uparrow, \text{array, int, char, num}$ |
| $\text{Tipo} \rightarrow \text{Simple}$ | 1 | int, char, num |
| $\text{Tipo} \rightarrow \uparrow \text{Simple}$ | 1 | \uparrow |
| $\text{Tipo} \rightarrow \text{array} [\text{Simple}] \text{ of Tipo}$ | 1 | array |
| $\text{Simple} \rightarrow \text{int}$ | 1 | int |
| $\text{Simple} \rightarrow \text{char}$ | 1 | char |
| $\text{Simple} \rightarrow \text{num}.. \text{num}$ | 1 | num |

Proc Main () // También podría llamarse "Proc S()"

Tipo ()

match (\$)

accept ();

endp

Proc Tipo ()

if (t in {int, char, num})

Simple ()

elseif (t in {↑})

match (↑)

Simple ()

elseif (t in {array})

match (array)

match (E)

Simple ()

match (J)

match (of)

Tipo ()

else

error

endp

Proc Simple ()

if (t in {int})

match (int)

elseif (t in {char})

match (char)

elseif (t in {num})

match (num)

match (..)

match (num)

else

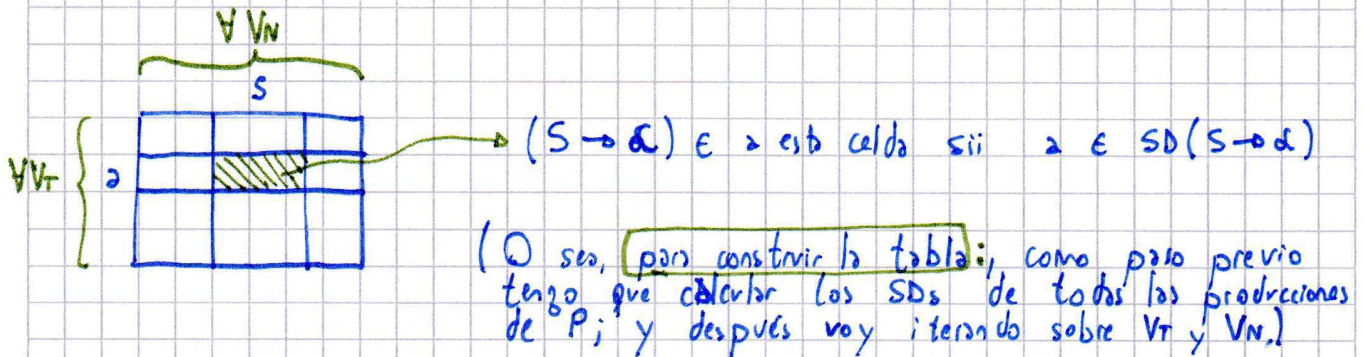
error

endp

Con los SD de 'Tipo' armo casos:
si el "token currently being pointed at"
(t) ∈ a el conj. de SD,
procedo a matchear los VT y
a invocar los procs. de los VN
(en el orden correspondiente); ~~XXXXXXXXXXXX~~
~~XXXXXXXXXXXX~~.

Analizador Sintáctico Desc. Predictivo con Tabla (y con pila :))

Se construye una tabla:



Una vez que la tabla está construida; algoritmo de parseo:

- Se inicializa la pila en $\begin{bmatrix} S \\ \# \end{bmatrix}$
- Sea "t" el símbolo apuntado en w
- M la tabla
- "tope" el símbolo en el tope de la pila

```

1 Repetir
2   if (tope ∈ V_T)
3     if (tope = t) {
4       desapilar tope
5       avanzar puntero 't' en 'w'
6     } else
7       devolver error
8   }
9   else // o sea: (tope ∈ V_N)
10    if (M, tope) = tope → x1...xn
11      desapilar tope
12      apilar xn, ..., x1 // si x1...xn = λ, no apila nada
13    else
14      devolver error // porque significa que no encuentro producción adecuada
15  fi
16 fi
17 Loop hasta que tope = #
18 devolver (error = t ≠ #)
  
```

Notar que con este algoritmo lo que va sucediendo es que en la pila siempre está apilado una FS.

Gramáticas LL(1)

• "LL" = left-left (se recorre de izq. a der, y expande el terminal más a la izquierda)

• "(1)" = cant. de símbolos hacia adelante que mira para saber que camino tomar.

Def de LL(1):

G es LL(1) sii $\left[\forall N \rightarrow \beta, N \rightarrow \beta' \in P; \text{ si } \beta \neq \beta' \Rightarrow SD(N \rightarrow \beta) \cap SD(N \rightarrow \beta') = \emptyset \right]$

→ (Q sea, dado un no-terminal N , y un terminal a , si G es LL(1) entonces existe a lo sumo una producción de la forma $N \rightarrow _$ tq a pertenece a los SDs de esa producción. No hay conflictos a la hora de elegir que producción usar para expandir en el algoritmo desc. predictivo (procedural o con tabla)). Solo las LL(1) pueden ser parseadas con un A.S. Desc. Predic.

Props:

Si G es ambigua \Rightarrow no es LL(1)

Si G es "reducido" y es LL(1) \Rightarrow no es recursiva a izquierdas.

(todas los símbolos alcanzables y activos, y sin símbolos anulares)
 un FS los genera una cadena de terminales
 contiene

A veces, dado una G que no es LL(1), es posible obtener una gramática equivalente que sea LL(1):

- reduciéndola
- eliminando recursión a izq. inmediata
- " símbolos anulares
- eliminando rec. a izquierdas de forma completa (la indirecta)

(Q sea: escribo la gramática con cuidado, le elimino la rec. izq. completa, y la factorizo entonces tal vez lo que quede sea LL(1).)

Eliminación de la recursión a izquierda inmediata

$$A \rightarrow \alpha_1 | \dots | \alpha_n | \beta_1 | \dots | \beta_m$$

se reemplaza por:

$$\begin{cases} A \rightarrow \beta_1 A' | \dots | \beta_m A' \\ A' \rightarrow \alpha_1 A' | \dots | \alpha_n A' | \lambda \end{cases}$$

Algoritmo de eliminación completa de la recursión a izquierda:Sea $G: \text{tq } A \rightarrow \lambda \notin P$ y $A \xrightarrow{*} A$ no es una derivación posible en G Sea $V_N = \{A_1, \dots, A_n\}$ 1 Para $i = 1 \dots n$ {2 Para $j = 1 \dots i-1$ {3 [Sustituir cada producción de la forma $A_i \rightarrow A_j \gamma$ por:

$$A_i \rightarrow \delta_1 \gamma | \dots | \delta_k \gamma ; \text{ donde } A_j \rightarrow \delta_1 | \dots | \delta_k$$

4 [Eliminar la rec. izq. inmediata de las nuevas producciones A_i (si las hubiere)

}

}

Factorización de gramáticas

$$A \rightarrow \alpha \beta_1 | \alpha \beta_2 \quad \text{puede escribirse como} \quad \begin{cases} A \rightarrow \alpha B \\ B \rightarrow \beta_1 | \beta_2 \end{cases}$$

AS Ascendentes

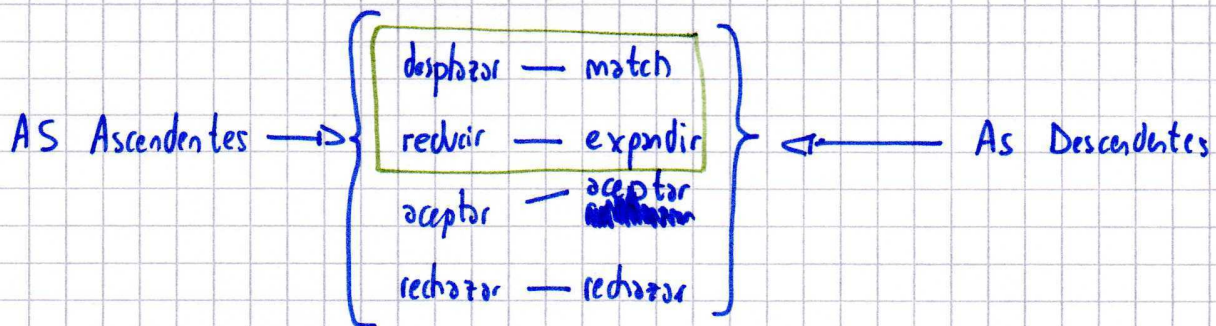
Los AS. Ascendentes (LR) son tal que =

- LR : left-to-right
right-most-derivation

- ~~Construyen~~ Construyen el árbol de parseo de abajo hacia arriba: de las hojas a la raíz

- Sus operaciones son:

| | |
|---|-----------------|
| { | desplazar * |
| | reducir * |
| | aceptar cadena |
| | rechazar cadena |

Comparación con los AS Desc:

• En los AS Desc., la operación de "expandir" consiste en dado un $N \in V_N$, y 't' el carácter apuntado, seleccionar una producción $(N \rightarrow \alpha) \in P$ y "expandir" N en α . ¿Cómo se seleccionaba la producción $(N \rightarrow \alpha)$? → Usando los SDs y viendo en cual conj. de SDs estaba incluido 't'

• En los AS Asc., la operación de 'reducir' consiste en, dado un 't' y sea p un "pivot" ("pivot": que aparece a el lado derecho de alguna prod.), busca la producción ' $N \rightarrow p$ ', y "reduzca" p por N .

DEF: Pivote: ("handle")

[Ser $\alpha p \beta$ una FSD, con $\beta \in V_t^*$, $\alpha \in (V_N \cup V_T)^*$, $p \in (V_N \cup V_T)^+$
 p es pivote sii $\exists (N \rightarrow p) \in P \wedge S \xRightarrow{*} \alpha N \beta \Rightarrow \alpha p \beta$

Alg. general AS Asc.

Sea: $w = a_1 \dots a_n$ cadena de entrada, seguida de una marca de fin de cadena ($\$$)
 tope: $(V_N \cup V_T)^+$ una ser. de caracteres en el tope de la pila
 t : símbolo apilado en la cadena de entrada

```

1. el Fin  $\leftarrow F$ 
2. error  $\leftarrow F$ 
3. repetir hasta el Fin
4.   case:
5.     tope es pivote:
6.       desapila pivote
7.       apila  $A$ , donde  $(A \rightarrow \text{tope}) \in P$ 
8.       pila =  $S \wedge$  fin de  $w$  ( $t = \$$ )
9.       el Fin  $\leftarrow V$ 
10.      aceptar  $w$ 
11.       $t \neq \$$ 
12.      apila  $t$ 
13.      desplaza  $t$  al sig. símbolo de  $w$ 
14.    else:
15.      error  $\leftarrow V$ 
16.      el Fin  $\leftarrow V$ 
17.  loop repetir
  
```

} **reducir**. La pregunta es:
 ¿Cómo determino si tope es pivote?
 }
 } aceptar
 } **desplazar**
 } rechazar

La cadena es aceptada cuando en la pila (u el tope) está S , y ya se consumió toda w

Parsers por precedencia

Se basa en relaciones de precedencia: \succ, \equiv, \prec , tg:

$$\succ, \equiv, \prec : (V_N \cup V_T) \times (V_N \cup V_T)$$

Defs intuitivas

$$V, R, S \in (V_N \cup V_T)$$

$R \succ S$ sii $\exists \alpha : \text{FSD} / \alpha = \dots RS \dots$, con R en el pivote y S no

$R \prec S$ sii $\exists \alpha : \text{FSD} / \alpha = \dots RS \dots$, con S en el pivote y R no

$R \equiv S$ sii $\exists \alpha : \text{FSD} / \alpha = \dots RS \dots$, con ambos, R y S, en el pivote.

→ ¿Cómo construyo la matriz de precedencia? (Algún algoritmo simple??)

Defs formales:

i) $X \prec Y$ si: $(A \rightarrow \alpha X B \beta) \in P$, y $B \xrightarrow{+} \gamma$

ii) $X \equiv Y$ si: $(A \rightarrow \alpha X Y \beta) \in P$

iii) $X \succ \alpha$ si: $(B \rightarrow \alpha C Y \beta) \in P$, y $C \xrightarrow{+} \gamma X$ y $Y \xrightarrow{+} \delta$ y $\alpha \in V_T$

Construir la tabla de precedencias:

① Cálculo P^+, U^+ y P^* para $(V_N \cup V_T)$:

a) $P^+(X) = \{Y \in V / X \xrightarrow{+} Y \alpha\}$

b) $U^+(X) = \{Y \in V / X \xrightarrow{+} \alpha Y\}$

c) $P^*(X) = V_T \cap (X \cup P^+(X))$

// $V = (V_N \cup V_T)$

Nota que si: $X \in V_T \Rightarrow P^+(X)$ y $U^+(X)$ son \emptyset

| V \ | P^+ | U^+ | P^* |
|--|-------|-------|-------|
| $V_T \left\{ \begin{array}{l} : \\ : \\ : \end{array} \right.$ | | | |
| $V_N \left\{ \begin{array}{l} : \\ : \\ : \end{array} \right.$ | | | |

- ② Calcular los pares (X, Y) consecutivos en las producciones (Todos derechos).
 $X, Y \in (V_N \cup V_T)$.
- ③ Con X, Y (con la colección de pares $X, Y \dots$) aplico el algoritmo para construir la tabla:

Para cada producción

Para cada par de símbolos contiguos (X, Y) // X indica la fila, Y la columna

Agrego a la tabla $X \equiv Y$

$\dots \dots \dots (X \in P^+(Y))$

$\dots \dots \dots (U^+(X) \supset P^*(Y))$

fin par

fin para

Agrego a la tabla $(\$ \in P^+(s))$

$\dots \dots \dots (U^+(s) \supset \$)$

Me queda una tabla:

| X \ Y | V_N | | | V_T | | | \$ |
|---------|-------|-----|-----|-------|-----|-----|----|
| | ... | ... | ... | ... | ... | ... | |
| V_N { | . | . | . | . | . | . | . |
| V_T { | . | . | . | . | . | . | . |
| \$ | . | . | . | . | . | . | . |

Si en alguna celda aparece más de una relación de precedencia, ahí tengo un conflicto!

Gramáticas propias:

- $\nexists A \rightarrow A, A \in VN$
- No hay simb. nulos (inactivos o inactivables)
- No hay prods λ (excepto $S \rightarrow \lambda$, y entonces S no está a la parte derecha de ninguna prod.)

Gramáticas de precedencia

G es de precedencia si es propia y entre cmlg. par de símbolos en $(VN \cup VT)$ hay a lo sumo una rel. de prec. definido

Gramáticas univocamente inversibles

Si $\forall (A \rightarrow \alpha), (B \rightarrow \beta) \in P; (\alpha = \beta \Rightarrow A = B)$

O sea: no hay 2 producciones con el mismo lado derecho. Ergo, a un parser asc. le resulta sencillo elegir la producción para reducir una vez que encuentre el pivote

Gramáticas de precedencia simple: si G es de precedencia, y univocamente inversible.

Algoritmo de AS. usando refs. de precedencia

Sea: t el token, tope (de un solo carácter) el tope de pila

Repetir

Si $\text{tope} > t$ // Hay un pivote en la pila!

$\alpha \leftarrow \lambda$

repetir

$t' \leftarrow \text{tope}$

$\alpha \leftarrow \alpha.t'$

desapila tope

hasta que $\text{tope} \leq t$

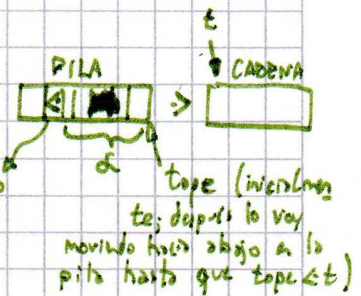
Si $\exists A : A \rightarrow \alpha \in P$

apilar A

Sino

devolver falso

Meto en α el pivote, y voy desapilando el pivote al mismo tiempo, moviendo 'tope' hacia abajo a la pila



si $\text{tope} < t \wedge \text{tope} \neq t$
devolver falso

Claro, porque si tope no es $<$, $=$, ni $>$ a t ... es que en cadena no ~~debe~~ debe ser aceptada

si $\text{pila} = \#S$ y $t = \#$
devolver verdadero

else

apilar t

avanzar pointer t en w

Loop repetir // hasta que se salga por retornar true o false

Obs:

. Pila se inicializa con $\#$

. Se acepta cadena cuando $\text{pila} = \begin{bmatrix} S \\ \# \end{bmatrix}$ y $t = \#$

. Para testear si $\alpha \in \mathcal{L}(s)$, se testea si $\alpha' = \alpha\# \in \mathcal{L}(s')$

. $\left[\begin{array}{l} \# \in x \quad \forall x: S \xrightarrow{+} x\alpha \\ y > \# \quad \forall y: S \xrightarrow{+} \alpha y \end{array} \right.$

- Pila inicializada a \$

PILA t CADENA

| | |
|----------------------------------|------------------|
| \$ | a c a c c b b \$ |
| \$ a | c a c c b b \$ |
| \$ < a < c | a c c b b \$ |
| \$ < a < c > | a c c b b \$ |
| \$ < a \equiv S | a c c b b \$ |
| \$ < a \equiv S < a | c c b b \$ |
| \$ < a \equiv S < a < c | c b b \$ |
| a \equiv S < a < c > | c b b \$ |
| S < a \equiv S < a \equiv S | c b b \$ |
| a \equiv S < a \equiv S < c | b b \$ |
| \equiv S < a \equiv S < c > | b b \$ |
| \equiv S < a \equiv S \equiv S | b b \$ |
| < a \equiv S \equiv S \equiv b | b \$ |
| a \equiv S \equiv S \equiv b > | b \$ |
| \$ < a \equiv S \equiv S | b \$ |
| S < a \equiv S \equiv S \equiv b | \$ |
| a \equiv S \equiv S \equiv b > | \$ |
| \$ S | \$ |

FIN

1 s

s

© compare 't' on 'tope' ©

r

s

s

© compare 't' on 'tope' ©

r

s

©

r

s

©

r

s

©

r

FIN

agrega t al tope de pila, desplazando a w

reemplaza el pivote en el tope de la pila por su reducción

③: no modifica más, es el acto de "darse cuenta" de que es el tope de la pila. HAY UN PIVOTE. Es condición previa al 'reduce'

Tratemos de deducir el algoritmo...

- ¿Cuándo reduce? → cuando ~~hay~~ a el tope de la pila hay un pivote
- ¿Cuándo desplazo? → " " " " " " no hay un pivote
- ¿Cómo empieza la pila? → con \$
- ¿Cuándo termina el algoritmo, exitosamente? → cuando en la pila queda $\begin{bmatrix} S \\ \$ \end{bmatrix}$ y t apunta a \$ (o lo análogo sin usar \$)
- ¿Cuándo termina SEGURO, exitosamente o no? → Cuando t apunta a \$.

Conds de error:

- El 'tope' y 't' no se hallan relacionados a la tabla.
- 'tope' $\geq t$ (lo que indicaría que hay un pivote a la tabla), pero no existe una producción en P con la que se pueda reducir.

Una de las cosas principales del algoritmo es esta: cuando $\text{tope} \geq t$, eso indica que en la pila hay un pivote, y ese pivote incluye al tope.

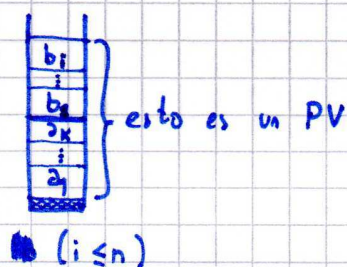
Pero el pivote son potencialmente VARIOS caracteres, entonces para REDUCIR hay que ser capaz de distinguir hasta donde 'desapilar' para reducir. Haciendo a mano es simple, se 've', se desapila hasta el ' \angle ' anterior al ' \geq '. A nivel algorítmico, se usa algo parecido.

Parsers LRAlgunas definiciones preliminaresPrefijo viable: (PV):

Si $\alpha\beta$ es una f.s.d.; digo que γ es P.V. si es un prefijo de $\alpha\beta$ (con $\alpha\beta \in (V_N \cup V_T)^*$)

Notar que: \triangleright el PV nunca se extiende más allá del último símbolo del pivote
 \triangleright un PV es todo lo que está en la pila de un A.S. Asc. shift-reduce:

Si: $\alpha = a_1 \dots a_k$, $\rho = b_1 \dots b_n$ y la pila tiene:



\triangleright El concepto de PV nos sirve para ver si vamos por el buen camino.

Item:

Si $(A \rightarrow \alpha) \in P$, con $\alpha = a_1 \dots a_n$ entonces la cadena ~~XXXXXX~~

$[A \rightarrow \alpha_1 \cdot \alpha_2]$ es un item; con $\alpha_1 = a_1 \dots a_k$ y $\alpha_2 = a_{k+1} \dots a_n$

La cantidad de items de una gramática es finita. Siempre

\rightarrow Un item representa: "estoy reconociendo esta producción (su lado derecho) y voy por acá (".")"

Item válido (IV) - asociado a un pref. viable:

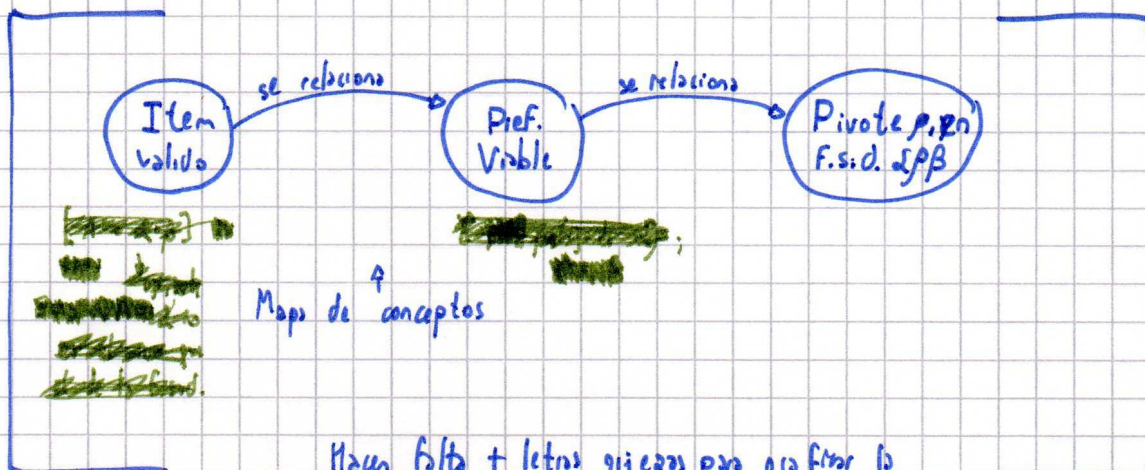
Dado un PV $\gamma = \delta\alpha$, decimos que $[A \rightarrow \alpha \cdot \beta]$ es un item válido para γ

sii

$$S \xRightarrow{*} \delta A w \xRightarrow{*} \delta \alpha \beta w$$

\downarrow
 $A \alpha$ es el PV

La def. de ítem válido sirve para ver que "vamos bien"



Hacen falta + letras griegas para graficar la situación:

F.s.d: $\alpha p \beta$; p es pivote

γ es PV de la f.s.d. Digamos $\gamma = \delta \Omega$

$[A \rightarrow \Omega \cdot Z]$ es ítem válido de $\gamma = \delta \Omega$ si:

$$S \xrightarrow{\gamma} \delta A \omega \Rightarrow \underbrace{\delta}_{\alpha} \underbrace{\Omega}_{p} \underbrace{Z \omega}_{\beta} \rightarrow \text{a la f.s.d. original}$$

Y la def. de I.V. nos dice "vamos bien, a cristo reconocemos Z vamos a poder reducir ΩZ por A "

Ítem completo: es de la forma $[A \rightarrow \beta \cdot]$ (o sea, indica que ya no falta reconocer nada para poder reducir).

AFND \rightarrow que reconoce los PVs de una gramática (y que dado un PV, puede decir cuáles son sus IVs asociados)

Dada $G = \langle V_N, V_T, P, S \rangle$, hajo:

$M = \langle K, (V_N \cup V_T), \delta, q_0, K \rangle$ donde:

// Toda los estados son finales

K : es el conj. de ítems de G , $\cup \{q_0\}$

δ :

i) $\delta(q_0, \lambda) = \{ [S \rightarrow \cdot \alpha] : (S \rightarrow \alpha) \in P \}$

ii) $\delta([A \rightarrow \alpha \cdot B \beta], \lambda) = \{ [B \rightarrow \cdot \omega] : (B \rightarrow \omega) \in P \}$

iii) $\delta([A \rightarrow \alpha \cdot X \beta], X) = \{ [A \rightarrow \alpha X \cdot \beta] \}$ // Vale igual tanto si X es V_N o V_T

Como un ítem representa "estoy reconociendo esta producción y voy por esa";
solo que M es un AFND \rightarrow , es directo para que pueda estar reconociendo más
de una producción al mismo tiempo; lo que equivale a decir que no sé
realmente que producción estoy reconociendo!

Ej. sea la fda $S \Rightarrow Sbc$ ($S' \rightarrow S \rightarrow SAc \rightarrow SaSbc$)
 \hookrightarrow pivote.

i.v. de S : $[A \rightarrow a.Sb]$ - iv. de Sa : $[A \rightarrow a.Sb]$ i.v. de SaS : $[A \rightarrow aS.b]$

iv. de $SaSb$: $[A \rightarrow aSb.]$ \rightarrow además de válido, es completo.

Def: (AFND- λ que reconoce los prefijos válidos de una gramática)

Sea $G = \langle V_N, V_T, P, S \rangle$, definimos $H = \langle K, \delta, q_0, K \rangle$ en donde:

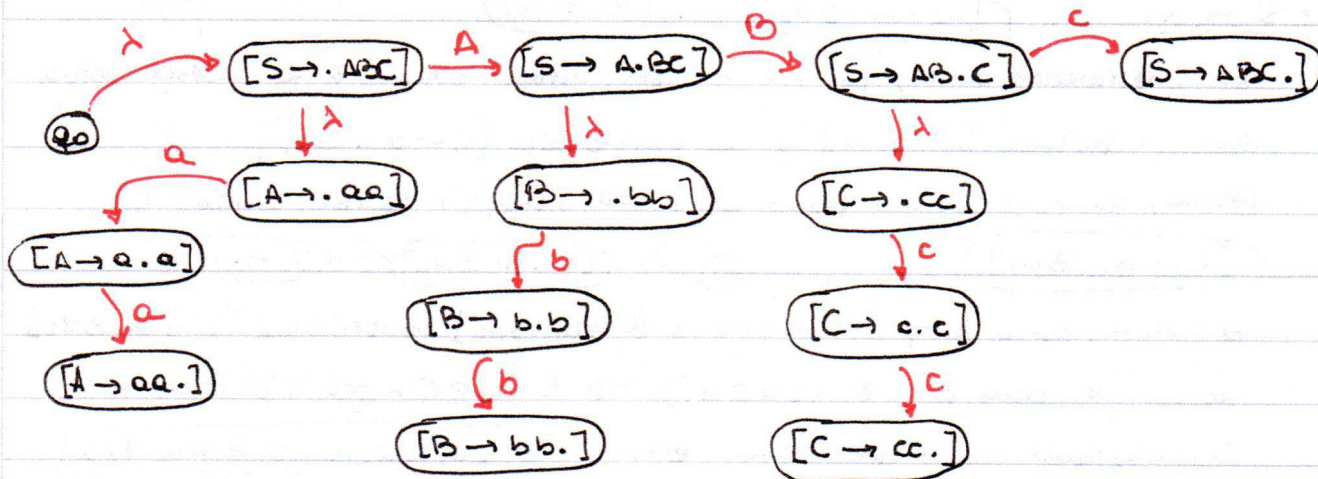
- K es el conj. de ítems de $G = \langle V_N, V_T, P, S \rangle \cup \{q_0\}$

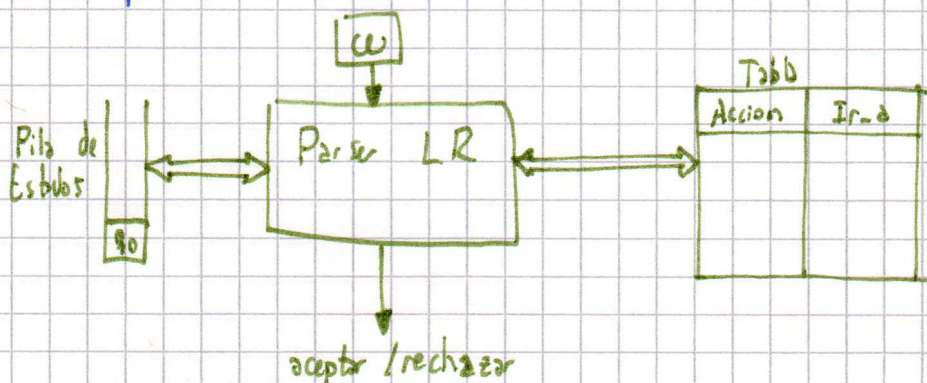
•

• δ es:

- $\delta(q_0, \lambda) = \{[S \rightarrow \cdot \alpha] : S \rightarrow \alpha \in P\}$
 - $\delta([A \rightarrow \alpha.B\beta], \lambda) = \{[B \rightarrow \cdot \gamma] : B \rightarrow \gamma \in P\}$
 - $\delta([A \rightarrow \alpha.x\beta], x) = \{[A \rightarrow \alpha x.\beta]\}$
- $\left. \begin{array}{l} x \in (V_N \cup V_T), \\ B, A \in V_N. \end{array} \right\}$

Por ejemplo: Sea G en $P = \{S \rightarrow ABC, A \rightarrow aa, B \rightarrow bb, C \rightarrow cc\}$



Parsers LRAlgoritmo general para parsers LR (no solo LR(0)) : t : símbolo a probar en w estado-tope: el tope de la pila (arranca con q_0 en el tope)

repetir indefinidamente

Según Acción [estado-tope, t]:• desplazar a q :apilar q avanzar t a prox. símbolo en w • reducir $A \rightarrow \beta$:desapilar $|\beta|$ veces $q \leftarrow \text{göto} [\text{estado-tope}, A]$ apilar q

• aceptar:

aceptar

Fin

• error:

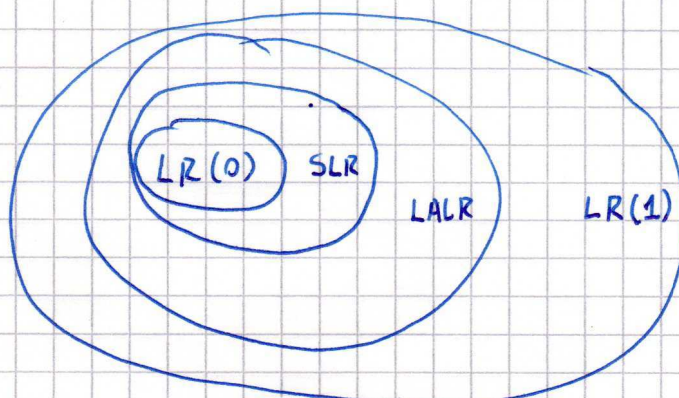
error / rechazar

Fin

Loop repetir

Es el mismo algoritmo para todos los parsers LR

Categorías de parser: En realidad, son categorías de gramáticas que son reconocidas sin conflictos por cada parser)



LR (K) :

- cont. de símbolos de entrada que uss para tomar decisiones de parseo
- construye una RMD, en reverso
- escribe la cadena "left-to-right"

- LR(0) y SLR → los más simples y menos costosos, pero más propensos a tener conflictos.
- ~~LR~~ LR(K) → los más complejos y costosos, pero menos conflictos
- LALR → algo intermedio

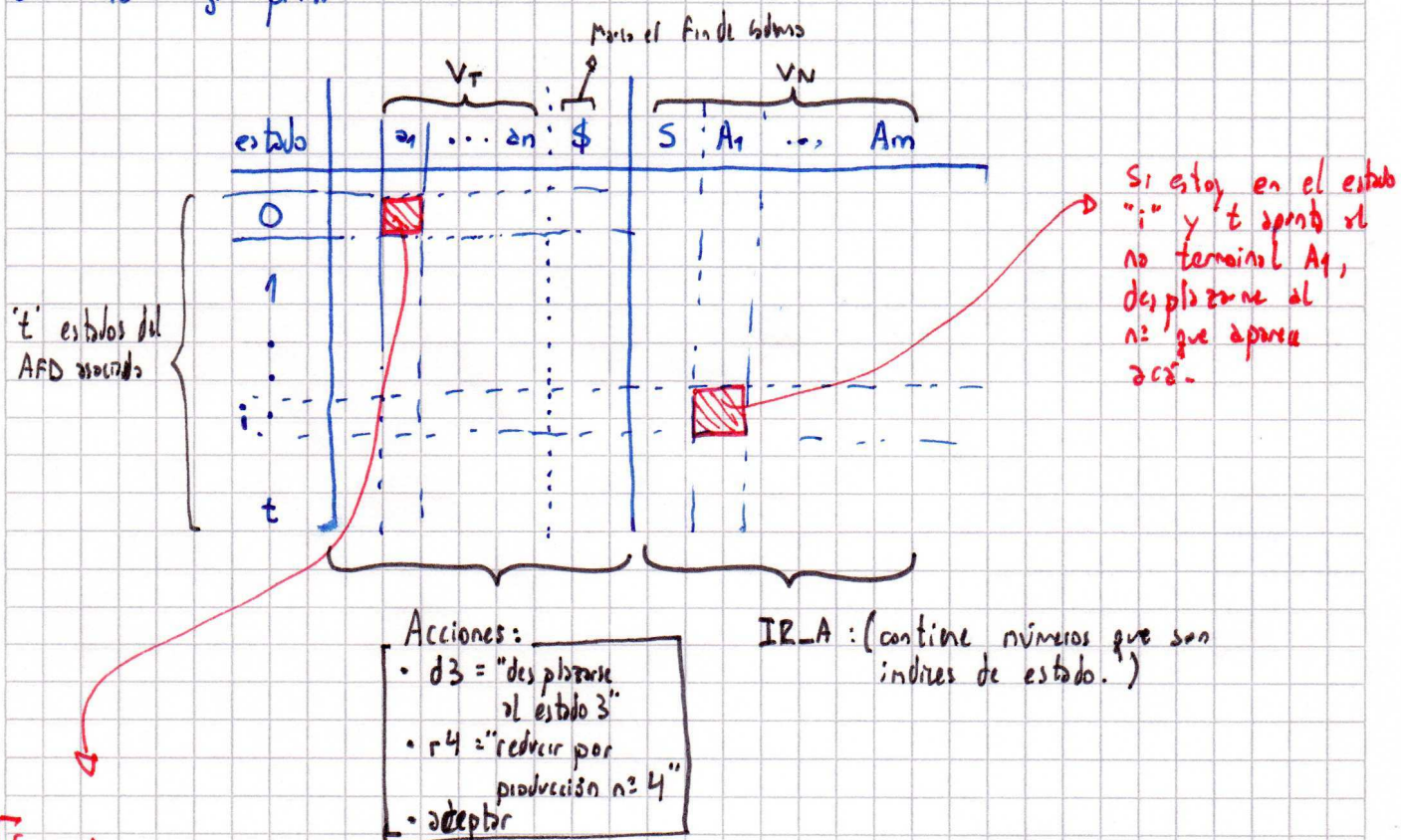
Todos estos parsers func. con el mismo algoritmo; pero varía cómo se construye la tabla "Acción/Ita". Para construir esta tabla es necesario antes construir un AFD (no AFND- λ), donde el estado de ese AFD va a tener un "núcleo" (un item) y la "clausura del núcleo" (un conj. de items relacionados al núcleo).

En el caso de las gramáticas LR(1), además de variar cómo se construye la tabla, también varía ~~construcción~~ la def. misma de "qué es un item" (el item incluye un carácter, en adición a lo que ya tenía).

Construcción de la tabla para un parser LR(0)

Hay 2 maneras de explicar esto: Una "algorítmica" donde se definen con algoritmos las funciones "Clausuras de Items", "goto", y tb. se definen con algoritmos la manera de construir el grafo y luego la tabla; y una más "hablada".

Para que pasemos un poco que estamos construyendo, una tabla "Acción/IR-A" tiene la sig. pinta:



Ejemplo: si acá aparece:

"d8" significa: si estoy en el estado 0 (q_0); desplazarme al estado 8 (q_8) por el símbolo a_1 (si t apunta a " a_1 ")

"r5" significa: si estoy en el estado 0 (q_0), y t apunta a " a_1 "; reducir por producción n° 5

Y si aparece más de una indicación \rightarrow conflicto (r-r / d-r)

Clausura de Items:

Sea el item $[A \rightarrow \cdot Bc]$, su clausura se calcula así:
me fijo ~~en el símbolo que le sigue al "."~~ en el símbolo que le sigue al ".".

- Si es un terminal, la clausura es \emptyset
- Si es un no terminal, (digamos 'B' como en este caso), agrego un item $[B \rightarrow \cdot \alpha]$ por cada producción $(B \rightarrow \alpha)$ en P ; y continúo:
para cada item que agregué a la clausura, le calculo a su vez su clausura y la agrego al conj. clausura del item original; y así hasta que ya no se pueda clausurar más (sigo hasta donde llegue)

Armando la tabla LR(0)

④ Armo el autómata LR(0)

El estado "q₀" o "I₀" es el que tiene en el núcleo a $(S' \rightarrow S \$)$. Creo I₀ un eso como núcleo y su clausura. y de ahí empiezo a "descubrir" otros estados.

Para cada estado me pregunto:

- ¿Por cuáles símbolos "se sale" de este estado? (V_n o V_r, no importa)
- son aquellos símbolos que aparecen a la derecha inmediata del "." en ese estado.
- Una vez que identifiqué eso, me pregunto: por el símbolo 'x', ¿en cuáles prods. avanzaría? esas prods. van a ser el núcleo del nuevo estado.

Ej:

| | |
|----------------|--|
| I ₀ | $S' \rightarrow \cdot S \$$ $S \rightarrow \cdot SA$ $S \rightarrow \cdot A$ $A \rightarrow \cdot (S)$ $A \rightarrow \cdot ($ |
|----------------|--|

de este estado se sale por los símbolos $\{S, A, (\}$

y para el símbolo S, se sale por 2 producciones/items: $S' \rightarrow \cdot S \$$ y $S \rightarrow \cdot SA$

así que el núcleo del estado al que se va por 'S' sería

| |
|--|
| $S' \rightarrow S \cdot \$$ $S \rightarrow S \cdot A$ |
|--|

Y así voy descubriendo/creando nuevos estados y las transiciones entre ellos.

② Armado de tabla LR(0)

1) Numero las producciones y los estados

2) Empiezo por I_0 y voy siguiendo el goto: ~~si~~ ^{a)} si de un estado cualquiera I_k salto por el terminal 'x' hacia el estado I_j , pongo en la coordenada $[I_k, x]$ la acción d_j . ^{b)} Si salto por el no-terminal A hacia el estado I_m , pongo el 'goto' en $[I_k, A] = m$.

^{c)} Si I_k tiene un ítem de la forma $[A \rightarrow \alpha.]$ ^{→ y $A \neq S'$} , ~~asociado a~~ la producción de índice "h" ($A \rightarrow \alpha$), pongo en Acción $[I_k, _]$ (para toda la fila de I_k) la acción "r h". (es decir, para cada $x \in V_T$, en esa fila pongo "r h")

^{d)} Si $[S' \rightarrow S.\$] \in I_k$; pongo "accept" en Acción $[I_k, \$]$

(a), b), c), d) son las cosas que hago de cada estado)

<Ver ejemplo en las hojas siguientes>

Notar una cosa acá: en la tabla LR(0), en el caso c); estoy marcando como "reducir por la producción número 'h'" a toda la fila (a la parte de la fila de ~~los~~ símbolos terminales); y precisamente ahí radica la propensión de este método a generar conflictos: podría tener que en alguna columna de esa misma fila, para algún terminal, tenía una acción "d_" o "r_" , dando así lugar al conflicto.

Precisamente, el método SLR apunta a solucionar parcialmente esto cambiando el criterio c): o sea, cambiando el criterio con el cual determino en CUALES columnas de la tabla acción voy a poner "r h". Para eso uso Siguiertes():

c)' (para SLR): Si I_k tiene un ítem de la forma $[A \rightarrow \alpha.]$ ^{→ y $A \neq S'$} asociado a la prod. número 'h' ($A \rightarrow \alpha$), pongo ~~en~~ Acción $[I_k, a] = r h$, sólo para aquellos $a \in V_T$ tales que $a \in \text{Siguiertes}(A)$.

Ejemplo de construcción de grafo y tabla LR(0)

FECHA

Ej:

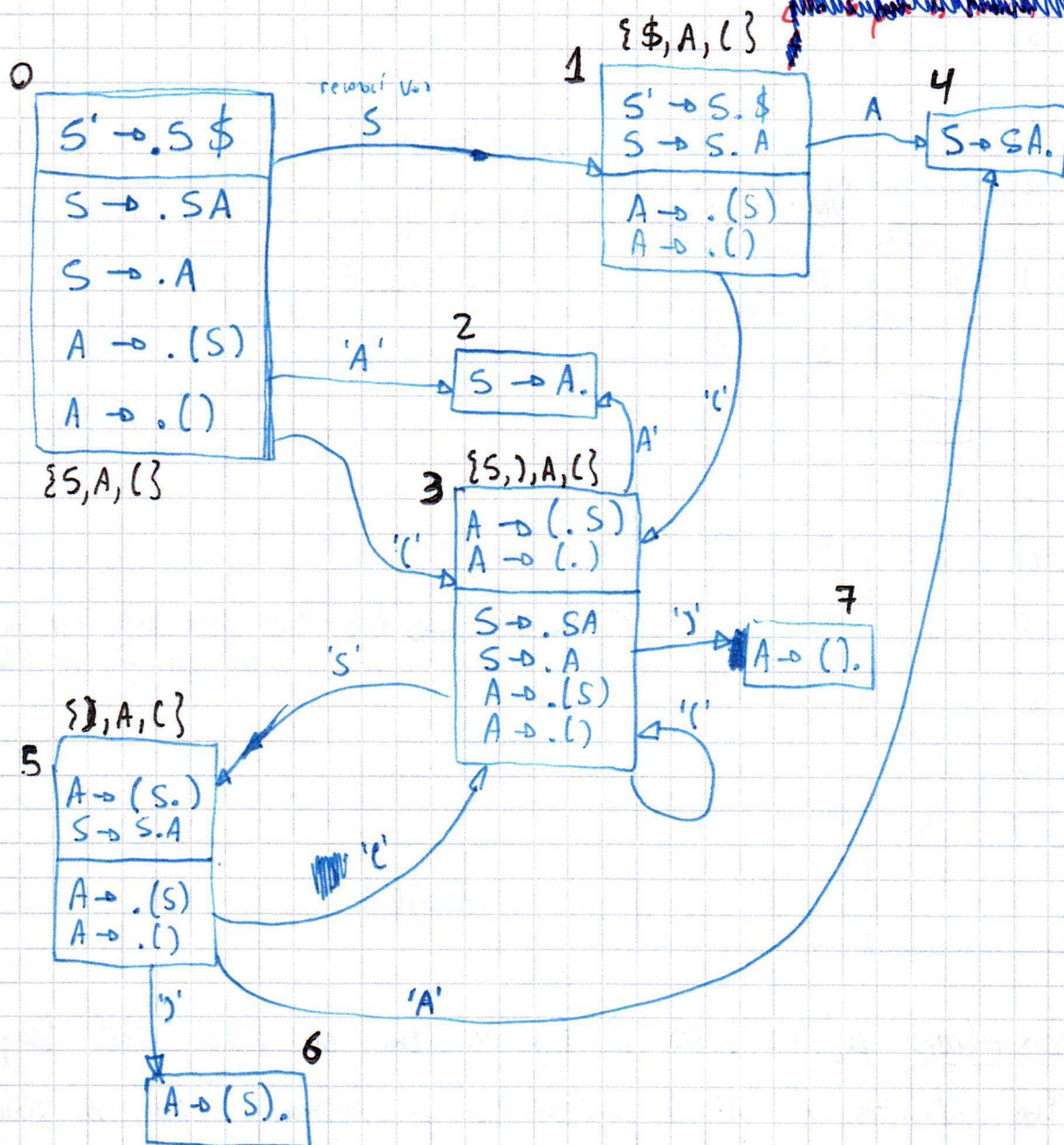
$S \rightarrow SA|A$

$A \rightarrow (S) | ()$

$S' \rightarrow S \$$

$L = \{(), ()(), ()(), \dots\}$

] Agregado, no estaba en la Original



LISTO

- Los estados con el \cdot al final (2, 7, 4, 6) no se desplazan a otros estados. Implican reducir por alguna producción. (reducen por la producción asociada a ese ítem)
- Para c/ estado pregunta: ¿Por cuáles símbolos me desplazo? ¿Y hacia qué estado me desplazo por tal símbolo?

Ahora que tengo el gráfico, puedo construir la tabla:

marca el fin de cadena

| estado | V_T | | | | V_N | |
|--------|-------|----|---------|--|--------------|---|
| | (|) | \$ | | S | A |
| 0 | d3 | | | | 1 | 2 |
| 1 | d3 | | aceptar | | 1 | 4 |
| 2 | r2 | r2 | r2 | | | |
| 3 | d3 | d7 | | | 5 | 2 |
| 4 | r1 | r1 | r1 | | | |
| 5 | d3 | d6 | | | | 4 |
| 6 | r3 | r3 | r3 | | | |
| 7 | r4 | r4 | r4 | | | |

Producciones por las que
puedo reducir:

① $S \rightarrow SA$

② $S \rightarrow A$

③ $A \rightarrow (S)$

④ $A \rightarrow ()$

ACCION

IR A

• ("d3" = "desplazarse al estado 3")

• ("r2" = "reduzco por la producción ②")

Mecanismo para reducir:

Para c/ estado que tenga un ítem con • al final, para ese ítem i identifico que n° de producción corresponde. Para ese estado y esa producción, para cada símbolo TERMINAL pongo "r_n", donde 'n' es el n° de producción.

Como ^{esta} ~~la~~ tabla no hay conflicto \Rightarrow puedo usar el parser LR(0)

¿Que es un conflicto?

- Si un mismo estado tengo más de un item con \cdot al final (conflicto reduce-reduce)
- Si para un estado, a la tabla, tengo que desplazarme (dn) y reducir (rm).

Seguimiento de cadenas en un parser LR(0)

Para parsear esto, uso la parte de "acción" de la tabla.

Ej: cadena: $((()))\$$

| | | |
|-----------------------|---|----------|
| | 0 | ((()))\$ |
| 0(3 | | ((()))\$ |
| 0(3(3 | |)())\$ |
| 0(3(3(3 | |)())\$ |
| 0(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3(3(3(3(3 | |)())\$ |
| 0(3(3(3(3(3(3(3(3(3(3 | |)())\$ |

0S1A4 \$

0S1 \$

} Listo, "accept"

Notar que a la tabla sólo necesitamos del lado izq. poner solo los estados no son necesarios los símbolos

SLR— Diferencias entre LR(0) y SLR + Ejemplo SLR (extraído de carpeta de Sole)

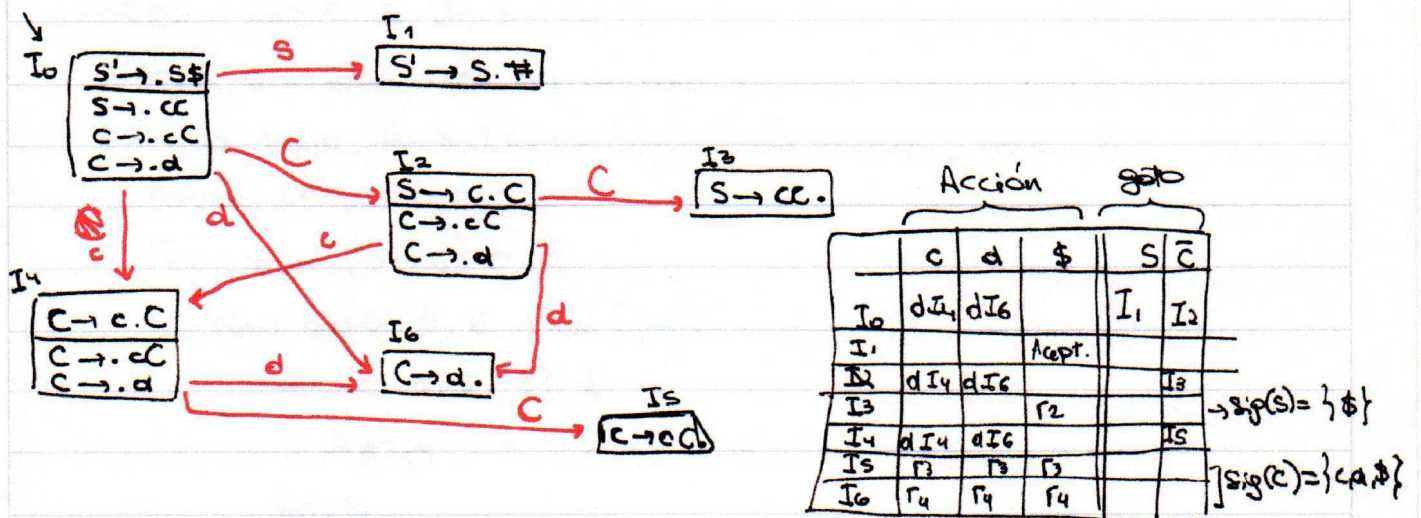
Tabla de acción LR(0)

$Acc[I, a] = \begin{cases} \text{desplazar } a \text{ a } j \text{ si } [A \rightarrow \alpha \cdot a \beta] \in I \text{ y } goto(I, a) = j \\ \text{reducir } A \rightarrow \alpha \text{ si } [A \rightarrow \alpha \cdot] \in I \text{ y } A \neq S' \text{ (En } S' \text{ no reducemos, sólo acepto o rechazamos)} \\ \text{aceptar si } [S' \rightarrow S \cdot] \in I \text{ y } a = \# \\ \text{error en otro caso} \end{cases}$

Tabla de acción SLR

$Acc[I, a] = \begin{cases} \text{desplazar } j \text{ si } [A \rightarrow \alpha \cdot a \beta] \in I \text{ y } goto(I, a) = j \\ \text{reducir } A \rightarrow \alpha \text{ si } [A \rightarrow \alpha \cdot] \in I \text{ y } A \neq S' \text{ y } a \in Sig(A) \\ \text{aceptar si } [S' \rightarrow S \cdot] \in I \text{ y } a = \# \\ \text{error si no} \end{cases}$

Ejemplo SLR: $G = \langle \{S', S, C, d\}, \{c, d, \$\}, \{S' \xrightarrow{(1)} S \$, S \xrightarrow{(2)} c, C \xrightarrow{(3)} c C, C \xrightarrow{(4)} d\}, S' \rangle$



| Pila | Cadena | Solución |
|--|--------------|----------------------------|
| \$I ₀ | c d c c d \$ | desplazar a I ₄ |
| \$I ₀ c I ₄ | d c c d \$ | desplazar a I ₆ |
| \$I ₀ c I ₄ d I ₆ | c c d \$ | reducir por 4: C → d |
| \$I ₀ c I ₄ C I ₅ | c c d \$ | reducir por 3: C → c C |
| \$I ₀ C I ₂ | c c d \$ | desplazar a I ₄ |
| \$I ₀ C I ₂ c I ₄ | c d \$ | desplazar a I ₆ |
| \$I ₀ C I ₂ c I ₄ c I ₄ | d \$ | desplazar a I ₆ |
| \$I ₀ C I ₂ c I ₄ c I ₄ d I ₆ | \$ | reducir por 4: C → d |
| \$I ₀ C I ₂ c I ₄ c I ₄ C I ₅ | \$ | reducir por 3: C → c C |
| \$I ₀ C I ₂ c I ₄ C I ₅ | \$ | reducir por 3: C → c C |
| \$I ₀ C I ₂ C I ₃ | \$ | reducir por 2: S → c C |
| \$I ₀ S I ₁ | \$ | Aceptor! |

LR(1)Def: Ítem LR(1) $[A \rightarrow \alpha \cdot \beta, t]$ es un ítem LR(1) si $(A \rightarrow \alpha \beta) \in P$, $t \in V_T$ Def: Ítem LR(1) válido $[A \rightarrow \alpha \cdot \beta, t]$ es un ítem válido LR(1) para $\delta = \delta \alpha$ prefijo viable si:
 $S \xRightarrow{*} \delta A w \Rightarrow \delta \alpha \beta w$ y $(t \in \text{Prim}(w)) \vee (w = \lambda, t = \#)$ Construcción de tabla de Pases LR(1)▲ Función Clausura LR(1)

• Entrada: cjbto I de ítems LR(1)

• Salida: cjbto C de ítems LR(1)

 $C \leftarrow I$ con elementos sin marcarMientras haya ítems $[A \rightarrow \alpha \cdot \beta, t]$ sin marcar y $\beta \in V_N$ Marcar $[A \rightarrow \alpha \cdot \beta, t]$ Para cada $(B \rightarrow \gamma) \in P$ Para cada $l \in \text{Prim}(\beta t)$ Si $[B \rightarrow \cdot \gamma, l] \notin C$ Agregar $[B \rightarrow \cdot \gamma, l]$ a C sin marcar

Fin si

Fin Para

Fin Para

Fin Mientras

Este ciclo interno adicional no estaba presente por LR(0)/SLR, y es la razón por la que el goto de LR(1) sea tan grande

▲ Función goto

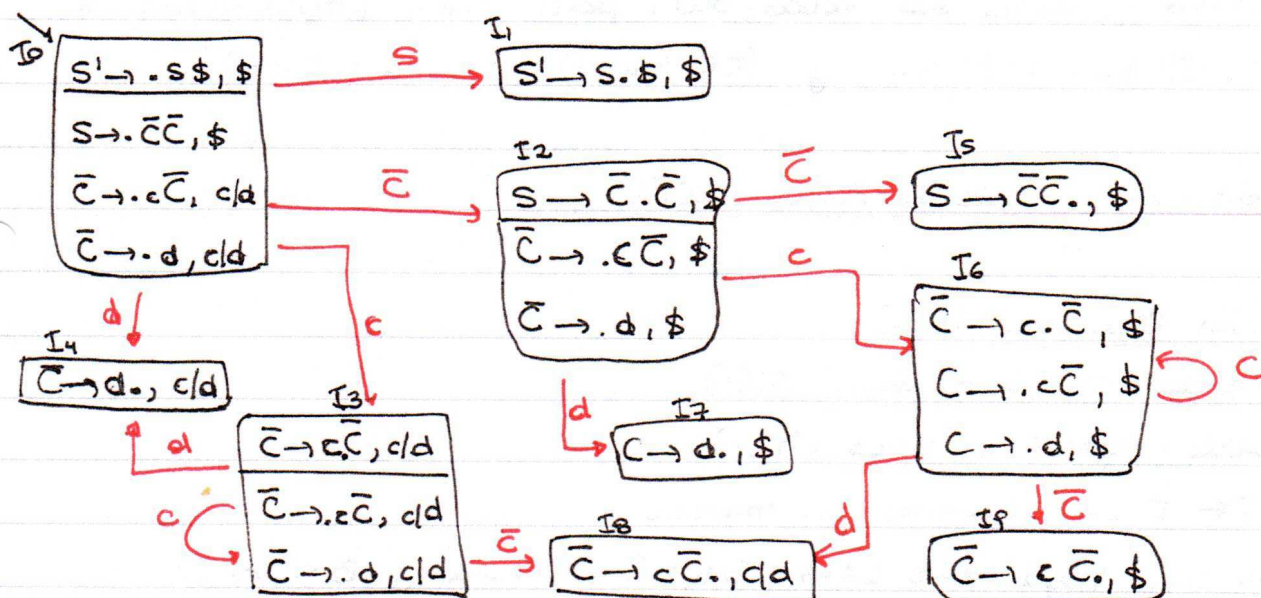
Retornar $\text{Clausura}_{LR(1)}(\{[A \rightarrow \alpha x \cdot \beta, t] : [A \rightarrow \alpha x \beta, t] \in I\})$

▲ Conj. de Ítems LR(1): Se construyen igual que para LR(0).

▲ Tabla de acción LR(1)

$$\text{Acción } [I, a] = \begin{cases} \text{desplazar } a \text{ a } j & \text{si } [A \rightarrow \alpha \cdot a \beta, b] \in I \text{ y } \text{goto}(I, a) = j \\ \text{reducir } A \rightarrow \alpha & \text{si } [A \rightarrow \alpha \cdot, a] \in I \text{ y } A \neq S' \\ \text{aceptar} & \text{si } [S' \rightarrow S \cdot, \#] \in I \\ \text{error} & \text{si no} \end{cases}$$

Ejemplo: G tal que $P = \langle S' \rightarrow S \$, \textcircled{1} S \rightarrow \bar{C} \bar{C}, \textcircled{2} \bar{C} \rightarrow c \bar{C}, \bar{C} \rightarrow d \textcircled{3} \rangle$



| | c | d | \$ | S | C |
|---|----|----|------|---|---|
| 0 | d3 | d4 | | 1 | 2 |
| 1 | | | Acpt | | |
| 2 | d6 | d7 | | | S |
| 3 | d3 | d4 | | | 0 |
| 4 | r3 | r3 | | | |
| 5 | | | r1 | | |
| 6 | d6 | d7 | | | 9 |
| 7 | | | r3 | | |
| 8 | r2 | r2 | | | |
| 9 | | | r2 | | |

| PILA | CADENA | lab |
|---------------------------------|---------|---|
| \$0 | cdccd\$ | despl. a 3 |
| \$0c3 | dccd\$ | despl. a 4 |
| \$0c3d4 | ccd\$ | reducir por 3: $\bar{c} \rightarrow d$ |
| \$0c3 \bar{c} 8 | ccd\$ | reducir por 2: $\bar{c} \rightarrow c\bar{c}$ |
| \$0 \bar{c} 2 | ccd\$ | desplazar a 6 |
| \$0 \bar{c} 2c6 | cd\$ | desplazar a 6 |
| \$0 \bar{c} 2c6e6 | d\$ | desplazar a 7 |
| \$0 \bar{c} 2c6c6d7 | \$ | reducir por 3: $\bar{c} \rightarrow d$ |
| \$0 \bar{c} 2c6c6 \bar{c} 9 | \$ | reducir por 2: $\bar{c} \rightarrow c\bar{c}$ |
| \$0 \bar{c} 2c6 \bar{c} 9 | \$ | reducir por 2: $\bar{c} \rightarrow c\bar{c}$ |
| \$0 \bar{c} 2 \bar{c} s | \$ | reducir por 1: $s \rightarrow \bar{c}\bar{c}$ |
| \$0s1 | \$ | Aceptar. |

Def: Núcleo de un ítem LR(1) ($I_1 \equiv I_2$)

I_1, I_2 tienen el mismo núcleo [redacted] \Rightarrow

$\forall A \rightarrow \alpha \cdot \beta \Rightarrow (\exists t : [A \rightarrow \alpha \cdot \beta, t] \in I_1 \Leftrightarrow \exists t' : [A \rightarrow \alpha \cdot \beta, t'] \in I_2)$

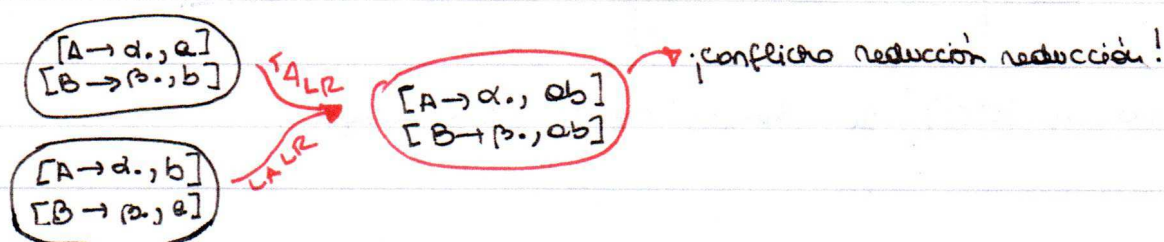
Ej: En el anterior $\Rightarrow I_4 \equiv I_7$

Construcción del AFD de un a.s. LALR

- Construir el AFD M LR(1) con K ciclos de estados
- Construir un nuevo ciclo de estados K' de la siguiente forma:
 - Si dos estados K_i y K_j son equivalentes (\equiv)
 - Para cada núcleo en K_i $[A \rightarrow \alpha \cdot \beta]$ el nuevo estado en K' es

$$\Delta \{ [A \rightarrow \alpha \cdot \beta, U] : U = U' \mid [A \rightarrow \alpha \cdot \beta, U'] \in K_i \vee K_j \}$$

Ej: de cómo LALR puede generar conflictos que no había en LR(1). Sup. que en el a.s. LR(1) tenemos



Atributo: variable tipada asoc. a un símbolo de la gramática

Regla de valoración^{"R.V."}: Es una función asoc. a una producción de la gramática, que relaciona a los atributos de los símbolos de esa producción:

$$E \rightarrow E + T \quad \{ E_1.val \leftarrow E_2.val + T.val \}$$

Atributo sintetizado:

Sea la prod. $A \rightarrow \alpha \in P$, $\alpha = \alpha_1.. \alpha_n$, $\alpha_i \in (V_N \cup V_T)$.

Sea el atrib. $A.a$ y la R.V. $A.a = f(\alpha_1.a_1, \dots, \alpha_n.a_n)$,

entonces $A.a$ es un atrib. sintetizado porque su valor sólo depende del valor de atributos en el lado derecho de la producción (y pg $A.a$ está del lado izq.)

Atributo heredado

$A \rightarrow \alpha \in P$, $\alpha = \alpha_1.. \alpha_n$, $\alpha_i \in (V_N \cup V_T)$. Si $\alpha_i.a_i = f(\alpha_1.a_1, \dots, \alpha_n.a_n, A.a)$

$\Rightarrow \alpha_i.a_i$ es un atrib. heredado porque su valor depende de ~~muchos~~ atributos asoc. a símbolos tanto en el lado izq. como derecho de la producción.

DDS: es un GLC con atribs. y R.V.s

"Def. dirigido por la sintaxis"

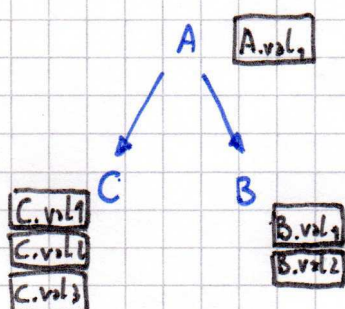
GA (Gram. de Atribs) es un DDS con RVs sin efectos colaterales (imprimir, actualizar una var. global, etc.)

Grafo de dependencias de atributos (GDA)

- Creo el A.D. de \mathcal{L}

- Agregando **NODOS**: Se agrega un **nodo de atrib.** asociado a un **nodo de símbolo** por cada atrib. de los símbolos que aparecen en el árbol.

La prod. $A \rightarrow CB$ genera una parte del árbol:



(Sup. que hay atrib. $A.val1, B.val1, \dots$ asociado a los símbolos A, B y C en la producción que generó esa parte del árbol.)

- Agregando **EJES**: Para cada **nodo de atrib.** v :

Para cada RV $v \leftarrow F(v_1, \dots, v_k)$, (tg esa RV esté asociada a la producción que creó esa parte del árbol); creo "K" arcos ~~desde v_i hacia v~~ desde v_i hacia v .

~~Como obtener una valoración de un árbol de derivación~~

Como obtener una valoración de un árbol de derivación

- Constr. el árbol de deriv. (AD)

- " el GDA

- Seguir el ordenamiento topológico y calcular valoración (si es posible)

↳ calcular primero los atrib. independientes, y después ir calculando de tal forma de antes de calcular el valor de un atrib., tener calculado los vals. de los atrib. de los que depende...

Ordenamiento topológico: dado un grafo dirigido acíclico, un ord. top. es cualq. ordenamiento n_1, \dots, n_k tg no existen aristas que vayan desde nodos que aparecen después en el ord. hacia nodos que aparecen antes ($\forall i, j; i < j \Rightarrow \nexists \text{ arista } n_j \rightarrow n_i$) // pero $n_i \rightarrow n_j$ "podría" existir.

Gramática S-atribuida : Sólo contiene atributos sintetizadores (un nodo ~~de~~ sólo depende de los "valores" ~~de sus hijos~~ sus "hijos")
Puede evaluarse empezando desde los hojas y yendo hacia la raíz.

Gramática L-atribuida :

Sus atribs son heredados o sintetizados. Pero en cualquier caso, si son heredados su valor sólo depende de atribs. asociados a:

- el nodo padre , y/o :
 - hermanos a su izquierda
- Ejemplo

Sea el atrib. heredado $\alpha_i.x_i$, tq:
$$\begin{cases} \text{Prod: } A \rightarrow \overline{\alpha_1 \dots \alpha_{i-1} \alpha_i \alpha_{i+1} \dots \alpha_n} \\ \text{RV: } \alpha_i.x_i = f(\alpha_1.x_1, \dots, \alpha_{i-1}.x_{i-1}, A.a) \end{cases}$$

entonces $\alpha_i.x_i$ depende sólo de sus hermanos a su izq. y de su padre,

obs: S-Atribuida \Rightarrow L-Atribuida (\nLeftarrow)

obs 2: Toda gramática L-Atribuida puede evaluar los atribs. de sus símbolos en una sola pasada:

Algoritmo visitar (b)

// sólo para L-atribuidas

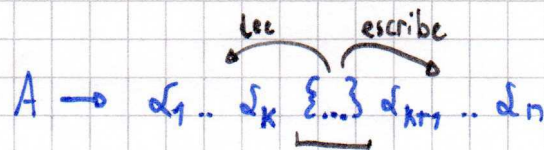
- 1 Para cada hijo a de b // de izq. a derecho
- 2 Evaluar atributos heredados de a ~~visitar(a)~~
- 3 visitar (a)
- 4 fin para
- 5 Evaluar los atributos sintetizados de b

TDS

Es una GLC donde se le asocia atribs a los símbolos de la gramática y se "insertan" acciones semánticas $\{ \}$ en lugares específicos de las producciones. Estas acciones semánticas son fragmentos de código; y a diferencia de los reglas de valoración tienen un orden de ejecución. El orden de evaluación es determinista.

Al igual que en las G.A. L-distribuidas, en los TDS los atribs. heredados sólo pueden depender del padre o de atribs. de símbolos a su izquierda en la producción.

¿Donde insertar los Bloq de código? $\{ \}$:



Un BC también puede tener acciones como imprimir algo en la salida

En el BC no debe ~~leer~~ **LEER** un valor de un atrib. de un símbolo a la derecha. Solo debe **LEER** de atribs de símbolos a la izquierda.

EL BC **ESCRIBE** en atribs de símbolos a la derecha. (Pero no me queda claro si puede **ESCRIBIR** tb. a símbolos a la izq o no...) Creo que puede escribir a cualquier lado...

A la hora de hacer el árbol se consideran a los BC como "hojas": Se recorre DFS de izq a derecha, se van evaluando los BC a medida que se van encontrando de este modo.

OSO, no estoy 100% seguro de esto.

TIPOS

Defs:

- Sistema de tipos: Adición de reglas para asignar expr. de tipos a las distintas partes del programa
- Un chequeador de tipos implementa un sistema de tipos
Los sist. de tipos se especifican de manera dirigida por la sintaxis.
- Sist. de tipos seguro: elimina la nec. de chequeo dinámico de errores de tipo.
- Leng. fuertemente tipado: el compilador garantiza que los programas que acepta se ejecutarán sin errores de tipo.
(igual algunos chequeos solo pueden ser hechos de manera dinámica.)
- Expresión de tipo: ("e.t.")
 - tipo básico
 - constructor sobre una expr. de tipo:
 - array $(\underbrace{I}_{\text{e.t.}}, \underbrace{t}_{\text{e.t.}})$ } Ej: $\text{int } a[10] : \text{array}(1..10, \text{integer})$
 conj. de índices
 - record: $(\text{nombre_campo}, \text{tipo}) \times \dots \times (\text{nombre_campo}, \text{tipo})$
 - apuntes: $\text{pointer}(\text{tipo})$
 - Funciones: $\underbrace{D}_{\text{tipo} \times \text{tipo} \times \dots \times \text{tipo}} \rightarrow \underbrace{R}_{\text{tipo} \times \dots \times \text{tipo}}$

• Equivalencia de tipos

◦ por nombre : si ambos son el mismo tipo básico o tienen el mismo nombre.

◦ por estructural : si ambos son el mismo tipo básico

◦ se forman aplicando las mismas construcciones de tipos.